UNIVERSITY OF OSLO
Department of Informatics

Regional Disaster Recovery Using Asynchronous Virtual Machine Replication

Amir Maqbool Ahmed

Network and System Administration

Oslo University College

June 5, 2010

Regional Disaster Recovery Using Asynchronous Virtual Machine Replication

Amir Maqbool Ahmed

Network and System Administration
Oslo University College

June 5, 2010

Abstract

Disaster recovery solutions for any system are complicated and in most cases expensive. Implementing DR solutions on service level is also not the best solution since it requires major efforts on each service. Remus, a recently included tool with Xen [1] can be used to implement DR solution on Virtual Machine level which will provide DR for all the services running on the VM. Here we will test Remus for regional distances where network speeds vary.

Acknowledgements

The author would like to thank all parties involved in this project without them it would not be possible to have completed this project. A special thanks to Gjøvik University College and their IT section for help in setting up the test system.

Contents

Chapter 1

Introduction

1.1 Motivation

The demand for service availability and dependency on IT services of modern societies have made Disaster Recovery(DR) and High Availability(HA) more important than ever. Some IT services are so critical that they are required to be up and running even after a natural disaster, like DNS, Email, Internet and access to public health records for hospitals and health personnel. Actually, IT communications and service availability becomes even more important in case of a disaster. This lesson has been learned the hard way after disasters like 2004 tsunami in Asia and hurricane Katrina in New Orleans[2].

If authorities are to help their citizens effectively and swiftly in case of a disaster, than emergency management systems must continue to function without any breakdown. In most developed countries health care, fire department and the law enforcing agencies are heavily dependent on IT communications and IT services, to be able to work efficiently. So there has to exist DR solutions for critical IT services, but DR solutions are expensive and complicated and therefore not affordable by many.

Traditionally, DR and HA has been expensive and hard, since redundant hardware and customized software is needed to construct such a solution. Naturally, different approaches exist to solve this challenge of giving a system ability to survive a failure. Replication of the whole system is one of the techniques used today with success but it is considered to be more expensive than application level check-pointing technique that replicates only the relevant data.

Classical solutions for DR are based on fail-over technologies where a master/slave environment is created. A master is operational under normal conditions while slave takes over operations when master fails. Although some services can be set up in a master/slave environment, like primary and secondary DNS, SMTP master/slave, not all services are designed this way and to implement a fail-over solution on the service level for all services will cost too much in terms of money and technical know how. Therefore these solutions are available to only a few.

1.2 Virtualization

A relative new trend in consolidating data-centers today is to use virtualization which allows better utilization of the hardware since several Virtual Machines(VMs) or instances of one or several different Operating Systems(OS) can run on the same hardware. The benefits and possibilities that virtualization offer has really opened eyes of system administrators around the world. The competition for creating the best virtualization system is tough and already there are several systems to choose from. For research purposes open source is a great resource available to researchers and one such open source virtualization system is Xen [1].

1.2.1 Virtualization and Disaster Recovery

Virtualization can offer a higher level of disaster recovery since it can be implemented on Virtual Machine(VM) level. Which means that all services running on the VM will automatically have DR capability. But a tool is needed to implement DR of VMs. Live migration can assist in implementing such solutions, but live migration alone cannot solve this since it requires graceful handover of VMs. It means that migration has to be initiated from the server which has the running VM and it will take some seconds(depending on the allocated VM memory) before receiving server has a running copy of the VM. In this short period, from initiation on primary until a running copy is available on secondary, both servers have to be up and running. In fact under live migration, handover is the most critical phase of the process.

What is needed is an environment where we have one running VM on a primary server and a true copy of it standing by on a secondary server and VM on the secondary server takes over operations *only* when either VM on primary or the primary server itself fails without warning, then we have achieved our goal and created an effective DR solution. It sounds simple enough but there are several challenges that have to be solved like suspending the backup VM on the secondary server and making sure that backup VM is an exact copy of the running VM at all times. This will introduce considerable overhead to the system and is the core of the problem.

1.2.2 Remus

A tool called Remus[3] is recently included in Xen which according to its developers will provide a way to implement High Availability(HA) or Disaster Recovery(DR) on running VMs in Xen environment. Remus takes the live migration feature of Xen to a higher level and takes full advantage of it to protect a VM. While live migration in Xen has shared storage as a requirement, Remus can protect a VM with and without shared storage therefore Remus has different levels of protection.

1. No disk replication: Only memory changes are sent to backup when the VM file system is located on a storage which is shared between the two

virtualized systems. If the primary VM dies in this mode, existing network connections to the VM will not survive and have to be reconnected.

2. Disk replication: In this mode disk changes are also sent to the backup along with the memory and existing network connections to the VM will also survive if primary VM or site is lost.

But since its new, Remus has not yet been tested for regionally separated systems. In this project a regional setup with Remus will be tested to find answers to questions like, is it possible to create a regional DR solution using Remus? What is the network cost(overhead) of running Remus on a VM in such an environment? How stable is it?

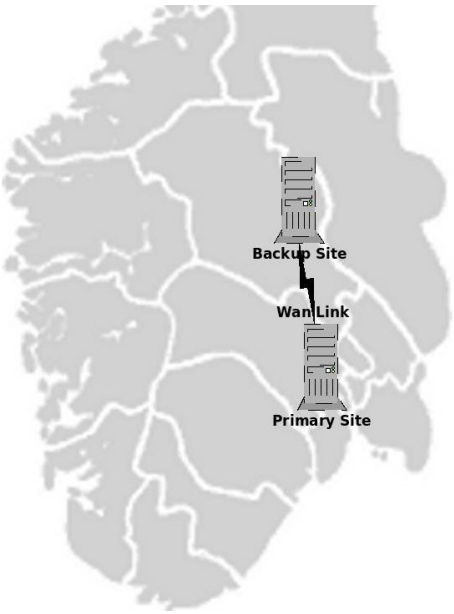

Figure 1.1: **Geographical Setup**

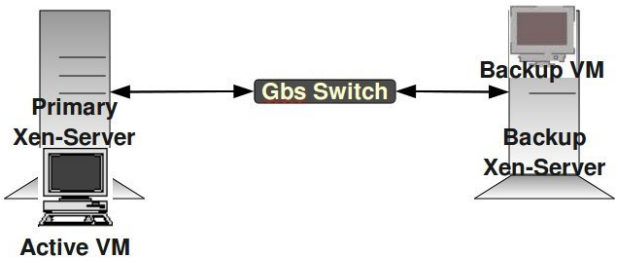

Figure 1.2: **Local Setup**

1.3 Problem statement

Ever increasing use of virtualized environments in data centers and server rooms along with high availability demand, demonstrates clearly a need for disaster recovery solutions for virtualized environments. But keeping an idle true copy of a running VM between sites separated by large distances, requires very high network speeds. For the project of testing Remus that has only been tested in local environments, the problem statement is:

Evaluate Remus, a recently included tool with Xen, for implementing disaster recovery solution for geographically separated VMs.

"Evaluate" here means following

1. Find out whether it is possible to run Remus between two geographically separated virtualized systems?
2. Run Remus with and without disk replication and measure network traffic created by this proses.
3. Observe stability of Remus.

All tasks mentioned above will be performed first between two local systems and then between two geographically separated systems for comparison. See figure 1.2.

Here "Geographically separated" means two different physical locations, preferably two different cities, see figure 1.1.

System setup and test scenarios

This project is implemented as a realistic system and none of the system services are simulated in a local environment. One primary and one secondary system is setup locally at the Oslo University College and another backup system is setup at the Gjøvik University College which is located approximately 120 KM from Oslo University College.

1.4 Thesis outline

This document is structured as follows: The background and related technologies are introduced in chapter 2. Chapter 3 lists testing scenarios and explains the methodology mainly used in this project. Chapter 4 are the results of the experiments while discussion and conclusions are presented in chapter 5 and 6 respectively.

Chapter 2

Background and literature

2.1 Disaster Recovery

Fail-over, or a disaster recovery plan typically involves deploying a remote fail-over scheme which allows a secondary server or site to take over mission-critical operations in the event of a disaster striking the primary site. What type of disaster recovery plan is needed and which technique should be used is heavily dependent on the IT structure of the site and services offered by it. Since implementing disaster recovery is an expensive undertaking, a thorough examination is needed before a solution is implemented.

2.1.1 Disaster Recovery or High Availability

Disaster Recovery(DR) and High Availability(HA) are terms often used in IT to indicate some sort of fault tolerance capability of a site. But there is a small difference between DR and HA. It can be said that DR grew out of HA capabilities such as clustering, but for DR, additional solutions were introduced to achieve for example clustering over metropolitan or global distances. But it does not mean that DR implies HA or that HA implies DR. For instance HA can be achieved from the same site using on-site clustering solutions. If the building is destroyed there is no other site to take over operations. For DR, if a service is broken on the primary in a way that heartbeat[4] continues to function properly, the fail-over of the service will not be initiated and backup server will not take over operations. To make a distinction it can be said that HA deals with hardware failure while DR deals with disasters where the whole site is lost[5]. The 1906 earthquake in San Francisco destroyed half the city and Federal building in Oklahoma city was bombed where data as well as backup was destroyed. Therefore if DR is to be effective there has to exist multiple geographically separated sites with same capabilities. These sites are normally referred to as primary and secondary or backup site, where primary site provides services until a disaster then the secondary takes over.

2.1.2 Solutions

Here is a brief mention of the techniques used to implement DR solutions but many of the concepts are the same for HA solutions as well, since DR can be viewed as an extension of HA. One major aspect of a DR solution is tolerable loss of data in event of an outage, we will look at methods that allow little or no loss of data. Generally we can say that most DR solutions involve data replication in such a way that least amount of data is lost and some logic exist to avoid data inconsistencies and manage services offered by the system.

Different levels of DR

As mentioned earlier, some services are designed with Disaster Recovery or High Availability in mind and can be set up with primary and backup nodes. DNS(Domain Name System) and SMTP(Simple Mail Transfer Protocol) are examples of such services and these are relatively easy to set up. This is called Service level DR solution since it is only the service that has DR capability. In this type of setup, only the service related data is replicated to the backup node and therefore the overhead is not very high. But more often than not sites offer different services and in most cases these services does not have DR or HA capabilities so another type of DR solution is needed to ensure business continuity.

In system level DR solutions the whole system or disk is replicated to the backup and therefore the overhead is much more compared to service level DR solution. In most system level DR solutions if primary server fails a backup server is booted at secondary site with access to the same data and services that the primary was running. Therefore services are unavailable until failure is detected and backup is up and running.

Different types of clusters are used in developing DR solutions and they are a popular choice.

Cluster types

Following cluster types exist for DR solutions

1. Stretched Cluster
2. Global Cluster

A *stretched cluster*[5] that spawns over several sites has its limitation in distance. For distances within three to four hundred KMs *stretched clustering* solutions are often used and this is possible today because of the network speed enhancements. For longer distances clustering technology has to be improved further.

Global cluster is another DR clustering technique where two different clusters are configured at separate sites and a management system decides where the application should run. The replication of data can be controlled from the cluster running the application or from the management system.

Replication methods

Replication methods can be grouped into three groups

1. OS-based
2. Storage-based
3. Application-based

DRBD (Distributed Replicated Block Device) provided by Linux is an OS-based replication solution. It is a software based, shared-nothing replicated storage that mirrors the content of block devices between servers. The block device is replicated in a master/slave mode and the master server has the full read/write access to the device whereas the slave server has no access but replicates all changes made by the master, silently, The replication can be asynchronous or synchronous.

IBM System Storage DS is an example of Storage-based replication here if the replication of one particular disk fails, then all disks within the containing consistency group are protected from further writes to ensure data consistency.

An example of application-based data replication is *IBM's DB2 HADR*, which requires a 2-node setup. One node acts as the primary, and all DB2 clients are connected to it. Updates to the local table on the primary node are replicated to the secondary, so that the secondary can update its local table.

These replication methods provides a method to preserve data but they do not provide a means to detect an system outage nor do they provide a means to deal with a system outage. For that we need additional logic built on top of these replication methods.

Following methods can be used to detect and deal with an outage at the primary site.

1. Scripting can be used to detect outage and make decisions about what to do.
2. DRBD along with Heartbeat can be used to make a responsive system.[6]
3. IBM's GDPS built on parallel Sysplex and data mirroring technologies, can be used to provide continuous application availability.

2.2 Virtualization

The idea of sharing computer resources is not new, in fact already in 1959 Christopher Strachey introduced "Time sharing" in computers[7]. This idea was popular and much discussed during 1960s and after succeeding in isolating application runtime environment IBM introduced virtualization as early as in late 1960's in their mainframe machines, but this technology has taken off only recently in other IT spheres.

In a distributed server environment a popular trend among system administrators have been to setup a new server for offering new services, this trend

had lead to massive expansion of the infrastructure. The reasons for putting up new servers varies, it can be compatibility of service with OS or service needing a special environment.

Virtualization's ability to consolidate servers to save power and better utilization of the hardware in an environmental focused world have really made this technology attractive for data-centers and other IT service providers with high electricity bills. They can now continue to run services on old OSes and the best part is that many different OSes can run on the same hardware sharing resources effectively.

Virtualization has evolved rapidly in the last decade and great deal of work have been done in improving performance, flexibility and management of Virtual Machines (VMs).[8].

2.2.1 Virtualization techniques

Three major virtualization techniques exist today

1. Full Virtualization
2. Operating System Virtualization
3. Hardware Virtualization (Paravirtualization)

Full virtualization

This technique uses full emulation of hardware resources, see figure 2.1. It is easy to setup and guest OS can be installed directly and used without modifications. But a disadvantage of this technique is 30% less performance than running directly on the hardware[9]. Most popular products using this technique are VMware Workstation, Virtual PC and VirtualBox.

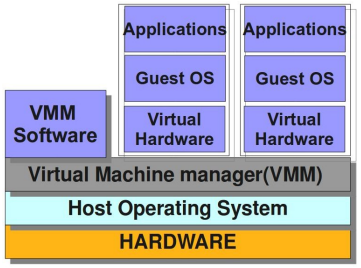

Figure 2.1: **Full Virtualization Technique**

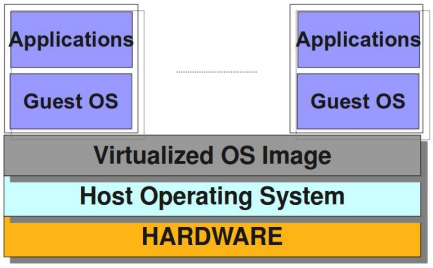

Figure 2.2: **OS Virtualization Technique**

OS Virtualization

In this technique, the host OS is virtualized and the guests are running parallel on their parent virtualized OS see figure 2.2. This technique provides very high performance almost the same as running on the physical hardware but

however the big disadvantage is that all virtual machines have to run the same OS image as the physical server. Products that use this techniques are Linux Vserver, OpenVZ and Solaris Containers.

Hardware virtualization (Paravirtualization)

Microprocessors(CPU) today have built in virtualization support that allows the host OS to expose hardware resources to guest OSes through an abstract software layer referred to as Virtual Machine Monitor(VMM) or a Hypervisor See figure 2.3. The guest OSes must be modified to communicate with the hardware through this hypervisor thus limiting deployment on legacy or proprietary OS.

The most popular products using this technique today are Xen, VMware ESX server and Microsoft HyperV. Since Xen has an open source version for academic use, it is the platform of choice for this project.

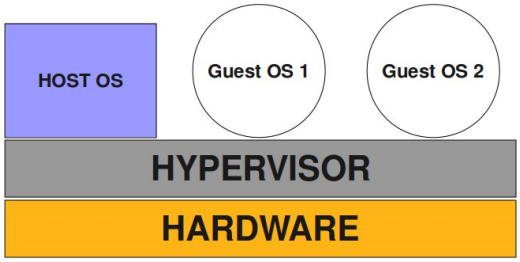

Figure 2.3: **Type 1 Hypervisor**

Live migration

A very special feature of hypervisors is *live Migration* which is the ability to migrate running VMs from one physical server to another.[10]. This capability can be used in new DR solutions and *is* actually used nowadays. Most hypervisors support migration of running VMs, Xen is also one of them.

2.3 Xen

Xen is a Virtual Machine Monitor(VMM) tool for the x86 architecture and falls into the paravirtualization(see fig. 2.3) category of virtualization techniques. It is an open source software, released under the GNU General Public License(GPL), and developed at the University of Cambridge. Detailed information about Xen can be found in "Xen and the art of virtualization"[11] paper.

VMs created in Xen are called Domains, and Domain0 is the host operating system which is responsible for controlling all the guest VMs. Domain0 therefore is the privileged Domain while VMs created are referred to as unprivileged Domains, DomU.

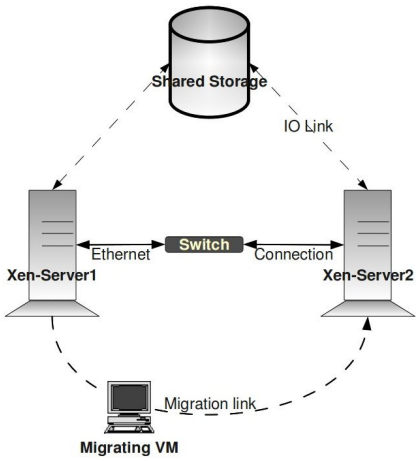

Figure 2.4: **Live migration of a running VM**

2.3.1 Live migration in Xen

Relocating running VMs to other Xen servers is called live migration and in Xen it has following requirements

1. Shared storage, where VM file system resides
2. Similar CPU architecture and identical features

In addition Xen servers that are involved in migration should share the same subnet segment, but it is not required.

2.4 Using live migration to develop Disaster Recovery Solution - Remus

As mentioned earlier live migration can be taken advantage of when creating new DR solutions and that is exactly what Remus[3] has done. To achieve a DR solution for a VM, a two system setup is needed where one is the primary system where running VM exists and the other is a backup system where a true copy of the protected VM exists which is in paused mode. See figure 2.5

2.4.1 How does Remus work

Remus takes advantage of the already existing live migration feature in Xen. To live migrate a VM, two Xen servers are setup, the VM is started on one server and its file system is shared by both the Xen servers. See figure 2.4. Migration is initiated from one of the servers and VM memory is transfered to the target server. When the transfer is complete, VM on the source server is destroyed while it is booted on the target server.

Remus initiates live migration of the VM and transfers VM memory to backup but instead of completing the migration, VM on the backup machine

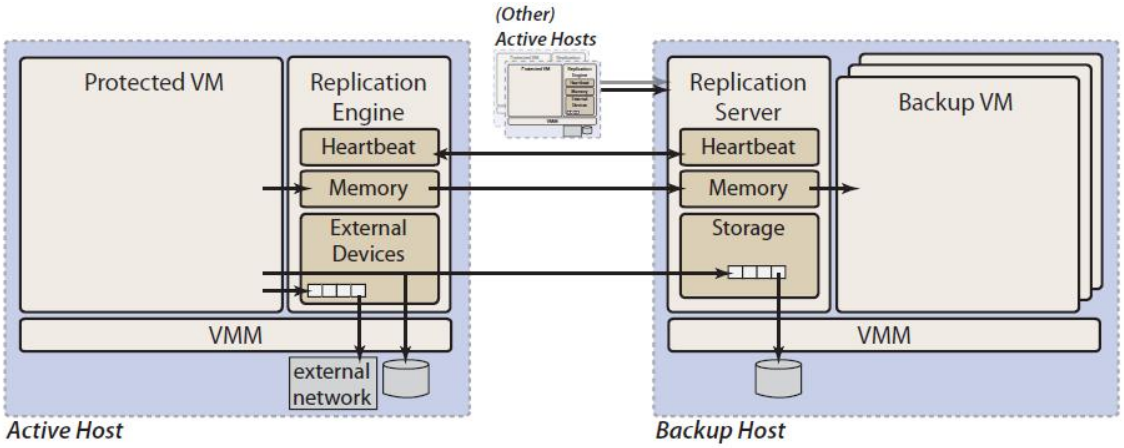

Figure 2.5: **Remus high-level Architecture**

is set to paused mode while primary VM continues its operation. All the subsequent changes on the primary VM from this point on are transmitted asynchronously to the paused VM at the backup server. This also works as the heartbeat of the system. If the backup VM doesn't get updates from the primary VM within certain amount of time, the backup VM mode is changed to running. Remus is run from Domain0 in Xen, see section 2.3 and can only protect one VM at any one time.

One requirement for live migration in Xen is that the VM file system is shared between source and target systems. But Remus is designed to work in two different modes:

1. Storage Replication mode.
2. Shared Storage mode.

Storage replication mode

In disk replication mode, VM storage is synchronized between primary and backup before VM is started and under startup of the VM that is to be protected, Remus opens an additional network channel through which all disk changes are replicated to the backup system.

This is done by setting the following command in the Xen configuration file for the VM in question.

```
disk = ['tap:remus:10.0.0.2:8000|aio:/dev/xen-domus/lenny-1,xvda1,w']
```

When the VM is started it will wait for Remus to open the disk replication channel. Following command will start Remus:

```
remus lenny-vm 10.0.0.2
```

Where lenny-vm is the name of the VM and 10.0.0.2 is the IP of the backup system. The port used for disk replication is 8000. The VM will now continue to load since the disk replication channel has been established. In this mode

the network connections will also be protected and will exist on the backup if primary was to fail. Its important that the VM disk exists on the same path on both primary and secondary systems, in this case */dev/xen-domus/lenny-1* which is a logical volume, and is synchronized before VM and Remus is started.

Shared storage mode

In this mode the VM disk is shared between primary and backup and there is no need to replicate disk changes. The command in VM configuration file is now:

```
disk = ['tap:aio:/mnt/nfs/var/nfs/lenny-1.ext2,xvda1,w']
```

Command for starting Remus is now:

```
remus --no-net lenny-vm 10.0.0.2
```

Where *–no-net* switch tells Remus that shared storage is used. Network connections in this case are not protected and will be lost if primary was to fail. Again the path to shared VM file system must be the same on primary and backup, in this case a VM file image at */mnt/nfs/var/nfs/lenny-1.ext2*.

Another feature of Remus is that the users can set the interval for updates sent to the backup.

```
remus -i 100 lenny-vm 10.0.0.2
```

Now the updates are sent to backup every 100 ms instead of 200 which is the default.

2.5 Virtual Private Network (VPN)

Virtual Private Network(VPN) is a technology used to establish and maintain a logical network connection. It is based on the idea of tunneling. Packets constructed in a specific VPN format are encapsulated within some other base or carrier protocol and de-encapsulated at the receiving side see figure 2.6. For VPN tunnels through Internet, packets of the VPN protocol used, are encapsulated within Internet Protocol(IP) packets. VPN protocols are many and support authentication and encryption to keep the tunnel secured. Examples of VPN protocols are L2TP, PPTP and IPSec but for this project an open source VPN network connection application called VTun is used.[12] . xcg

VTun

According to the developer of VTun

> *The easiest way to create Virtual Tunnels over TCP/IP networks with traffic shaping, compression and encryption.*

VTun is setup in a client-server environment and a single connection is established when client connects to a specified port on the server. A virtual TAP device is created on both nodes as endpoints for the connection therefore both nodes share the same Local Area Network(LAN) security.

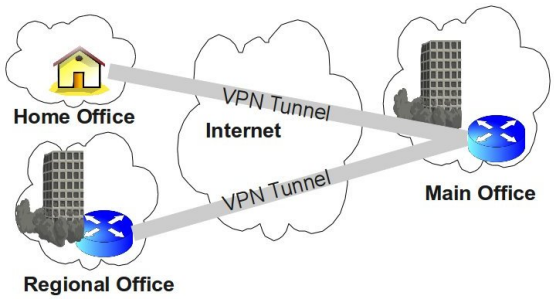

Figure 2.6: **VPN Model**

Chapter 3

Methodology

This chapter describes the topology design and its implementation. Challenges facing setting up a centralized network design as well as storage setup solutions used are also covered here along with techniques used for measuring network traffic. Testing scenarios are then covered in another section at the end of this chapter.

3.1 Objectives

Based on the concepts introduced in the background chapter, the problem statements discussed in section 1.3, are reiterated here in a more formal manner.

- Setup the experiment architecture.
- Conduct experiments while running the system and measure network traffic created.
- Analyze results.

The most important part of this project is measuring network traffic created by Remus in different modes for regional distances. The system setup is configured for that in focus. Network traffic is the main overhead of running a system like Remus and hence is important.

3.2 Environment

This project is based on cooperation between two parties

- Oslo University College, Norway
- Gjøvik University College, Norway

A regional networking infrastructure needs to be setup in order to connect the two regional servers together. The network connections to outside world for both Colleges are provided by the same Internet Service Provider

Alias	Host	Location	CPU	Memory	Network card(s)
Primary	remus1	Oslo	Intel® Dual Core, 2.4GHz	1GB (DDR 667MHz, 1.5ns)	**Eth0**: NetXtreme BCM5754 Gigabit Ethernet PCI Express **Eth1**: D-Link DGE-528T Gigabit Ethernet Adapter
Oslo	remus2	Oslo	Intel® Dual Core, 2.4GHz	1GB (DDR 667MHz, 1.5ns)	**Eth0**: NetXtreme BCM5754 Gigabit Ethernet PCI Express **Eth1**: D-Link DGE-528T Gigabit Ethernet Adapter
Gjøvik	remus3	Oslo	Intel® Pentium® 4 2.8GHz	1GB (DDR 533MHz, 1.9ns)	**Eth0**: NetXtreme BCM5751 Gigabit Ethernet PCI Express

Table 3.1: Xen Servers Hardware Information

Software	Version	Function
OS	Linux 2.6.18.8-xen	Linux Kernel image for Xen
VTun	3.0.2-1.1	Virtual tunnel over TCP/IP networks
Xen	Xen 4.0.0-rc3-pre	Virtual Machine Monitor(VMM)
httperf	0.9.0	HTTP performance measurement tool
openssh-server	1:5.1p1-5	Secure shell server
openssh-client	1:5.1p1-5	Secure shell client

Table 3.2: Xen Servers Software and Operating System

Device	Vendor	Type	Description
Switch	Zyxel	gs-105b Gigabit Switch	Connecting eth1 on Primary to eth1 on Oslo
Switch	D-link	DGS-1005D Gigabit Switch	Connecting Eth0 on Primary to Oslo College's public network

Table 3.3: External Hardware Information

which is an advantage regarding bandwidth available from Oslo to Gjøvik. The hardware is provided by Oslo University College while Gjøvik is hosting the regional backup server with gigabit network, allowing incoming connections only from primary server located in Oslo.

3.2.1 Physical servers and external hardware

As mentioned in section 2.4.1, Remus utilizes live migration feature of Xen which requires shared storage and similar CPU architecture, all servers have similar hardware, software configuration and operating systems. Table 3.1 shows hardware information for the servers used at different locations while software environment for all servers is shown in table 3.2. Aliases used will be the main naming references throughout the paper. All three physical servers are used as Virtualization servers, Primary server is additionally used as Network File System(NFS) server as well as virtual tunnel server. Primary and Oslo servers have two ethernet interfaces (eth0 and eth1) while Gjøvik has one

Alias	Host	OS	VCPU	Memory
VM	lenny-vm	Debian Lenny 5.0	1	512MB

Table 3.4: Virtual Machine Hardware Information

Software	Version	Description
bonnie++	1.03d	Hard drive performance testing tool
lighttpd	1.4.19	Webserver with minimum memory footprint
openssh-server	1:5.1p1-5	Secure Shell Server

Table 3.5: Virtual Machine software Information

see table 3.1. Primary and Oslo servers are connected together using their eth1 interfaces through a Gigabit switch which makes it an isolated back-net and has no other traffic. While another Gigabit switch is used to connect eth0 on Primary and eth0 on Oslo to Oslo Colleges public network which also provides Internet access see 3.3. Gjøvik server is connected to Gjøvik Colleges public network through eth0 interface directly.

3.2.2 Virtual Machine Instance

For this project, one instance of a virtual machine is used with hardware configuration shown in table 3.4 and software configuration shown in 3.5. The VM creation is described in section 4.1.3.

3.2.3 Infrastructure Design Challenges

Different technologies are involved in this project and their dependencies must be satisfied in the regional infrastructure. This increases the difficulty of setting up the network and storage infrastructure. Network has to be centralized for migration and storage has both centralized and independent setup for testing different Remus modes.

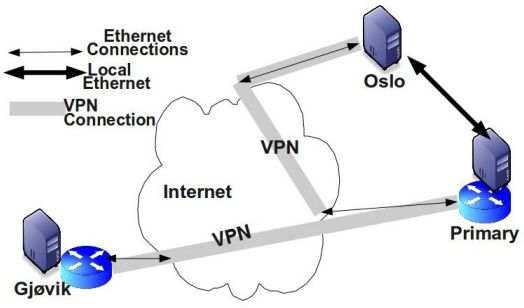

Figure 3.1: **Setup Overview**

Alias	Eth0	Eth1	Virtual Bridge	Description
Primary	128.39.75.154/23	10.0.0.1/24	192.168.0.1/24	Primary virtual node VPN server, Shared Storage server
Oslo	128.39.75.199/23	10.0.0.2/24	192.168.0.2/24	Local Backup virtual node
Gjøvik	128.39.80.80/22	X	192.168.0.3/24	Regional Backup virtual node

Table 3.6: Network Configuration

3.2.4 Network Infrastructure

Primary and Oslo servers have two physical interfaces, one public and one private interface. Private interfaces of Primary and Oslo are used for Remus network traffic only. Public interface on Primary is used for Remus traffic to Gjøvik and VPN traffic to both Gjøvik and Oslo. While public interface on Oslo is only used for VPN traffic. Gjøvik has one public interface for Remus and VPN, see figure 3.1. Since Gjøvik has different public subnet than Primary, special network configuration is needed for VMs to be accessible from Primary server. To solve this, a private virtual network is created for VMs and a password protected VPN tunnel is established to connect these networks. VMs private network traffic must be routed correctly for Internet access but it is not needed for this project since tests are conducted from Primary server. Along with connecting private network segments, VPN is also used for shared storage traffic (NFS).

3.2.5 Storage

Remus has two working modes see section 2.4.1. for testing these modes shared storage and Logical Volumes are used on all servers. Primary is running a NFS server sharing the VM filesystem images. Oslo and Gjøvik uses VPN tunnel to connect to NFS server on Primary. For testing Disk replication mode, a logical volume with VM's filesystem was created on each server and synchronized before it was used.

3.2.6 Traffic Measuring Techniques

A technique for measuring traffic has to be implemented to isolate different traffic types from other network traffic. For this project iptables [13] rules are used to filter Remus, http and SSH traffic. Because more often than not, a server in a normal production environment will use a firewall for protection. For each traffic type two iptables rules are created, one for outgoing and one for incoming traffic but only one direction is used f.ex. for Remus, outgoing traffic to backups from Domain0 on the Primary is used as basis for results. Traffic in other direction is mostly acknowledgements.

3.2.7 Logging

Iptables reports packets and bytes for traffic that matches a rule. This information is logged to a file for both incoming and outgoing traffic along with the

time, using a perl script. Logging is done every second for duration of a test. An example of logging traffic for live migration of VM follows:

Time	TX Packets	RX Packets	TX Bytes	RX Bytes
1271878912	0	0	0	0
1271878918	4512	1864	6739477	96952
1271878919	10995	3097	16463337	161068
1271878920	46195	12282	69260229	649428
1271878921	72052	16877	108044645	888368
1271878922	101026	23255	151501089	1220024
1271878923	126182	29719	189235089	1556152
1271878924	144501	32590	216713589	1705444
1271878925	163859	36568	245749149	1912300
1271878926	191798	43351	287653745	2265016
1271878927	216899	49541	325303221	2586896
1271878928	259192	61217	388734617	3194048
1271878929	302979	72574	454406581	3784612
1271878930	354782	88373	532092033	4606160
1271878931	372436	92962	558565857	4844788

For logging disk activity in the VM, a tool included in Xen called *Xentop* [14] is used. Xentop displays real time information about running VMs on a Xen virtualization system. It can run in different modes. For parsing information reported by Xentop from a script, it can be run in batch mode. For virtual block devices, Xentop will report total reads, sectors read, total writes and sectors written by a VM.

3.3 Testing Scenarios

In order to measure network traffic created by running Remus in different modes (see section 2.4.1), various test scenarios are designed. All scenarios are run from Primary for a virtual machine described in table 3.4. First, Oslo is used as backup and then same scenario is repeated using Gjøvik as the backup node.

Different traffic types are involved in each test and it is important to make distinction between them along with their directions. In general, all traffic from Domain0 on Primary to backups is referred to as TX traffic while all traffic from Domain0 on Primary to the protected VM is referred to as RX traffic seen from the VM perspective. For converting bytes to kilobytes and megabytes

$$bytes/1024$$

and

$$bytes/1048576$$

are used respectively.

The description of different activities performed towards the test VM for test scenarios 3.3.3 and 3.3.4 is given under. Two types of activities are performed towards the test VM from Domian0 on Primary:

1. A webserver performance tool called httperf [15] is used to generate http network traffic

2. A 100 MB file is transfered using ssh

The following command is used to run httperf benchmarking tool:

```
httperf --hog --server 192.168.0.5 --num-conn 10000 --ra 100 --timeout 5
```

Using with these options over, the httperf test will run for 100 seconds generating 100 http connections to the web server running on the test VM asking for the default index.html file. This file in the VM is the default web page installed when lighttpd web server is installed and is about 3.5 kilobyte big.

The ssh file transfer is started running following command:

```
scp test.file root@192.168.0.5:/root/
```

The command over will transfer the *test.file* to the test VM and save it in the */root* directory. The *test.file* is exactly 100MB big.

```
remus1:/root# ls -al test.file
-rw-r--r-- 1 root root 100000000 2010-03-21 01:46 test.file
```

3.3.1 Scenario 1: Measuring Live migration traffic

In this scenario, network traffic created by a live migration process in Xen is measured. Xen configuration file for the VM is used to start the VM and live migration is initiated while the traffic measuring script is running. The VM's filesystem is shared between the servers. This test is repeated 20 times.

3.3.2 Scenario 2: Measuring traffic running Remus with no disk replication, Idle VM

For comparison with live migration data, a pure Remus run on a VM is conducted and its traffic recorded. The VM is started with the same Xen configuration file as in 3.3.1 and Remus is started fro the VM while network traffic is logged. It is important that no operations are performed on the VM since that will create additional traffic. The VM filesystem is shared and this scenario is repeated 10 times.

3.3.3 Scenario 3: Measuring traffic running Remus with no disk replication, SSH and Web tests towards the VM

The intension here is to measure network traffic created by performing different tasks on the VM. The description of these tasks is given in section 3.3. Which means that after startup, the VM is no longer idle but a web performance tool httperf is run to test the performance of the web server running on the VM. While this test is running, an exactly 100 MB file is transfered to the VM repeated times.

Following traffic types are logged in this scenario:

- Remus update traffic to the backup
- Http traffic to the VM
- ssh traffic to the VM

Test description(Measure)	Backup	Storage type	Frequency
Live migration traffic	Oslo	Shared	20 times
Live migration traffic	Gjøvik	Shared	20 times
Remus traffic on idle VM	Oslo	Shared	10 times
Remus traffic on idle VM	Gjøvik	Shared	10 times
Remus, http and ssh traffic while running tests on VM	Oslo	Shared	once
Remus, http and ssh traffic while running tests on VM	Gjøvik	Shared	once
Remus, http, ssh and disk traffic while running tests on VM	Oslo	Replicated	once
Remus, http, ssh and disk traffic while running tests on VM	Gjøvik	Replicated	once
Remus, disk TX traffic, running disk benchmark from VM	Gjøvik	Replicated	once

Table 3.7: Summary of test scenarios

3.3.4 Scenario 4: Measuring traffic running Remus with disk replication, SSH and Web tests towards the VM

This scenario is same as scenario 3 in 3.3.3 but now the VM disk is not shared any more. VM disk is replicated through a communication channel created by Remus therefore this traffic is in addition to the Remus update traffic to backup. The following traffic types are logged:

- Remus update traffic to the backup
- Http traffic to the VM
- ssh traffic to the VM
- Disk update traffic to the backup

3.3.5 Scenario 5: Measuring traffic running Remus with disk replication, running bonnie++, regional only

In this scenario only regional test will be conducted to find out how well Remus can handle extensive disk activity. Test is setup as follows: The test VM is started with disk replication to Gjøvik. It will start and wait for Remus to start the disk replication before startup is complete. Remus is started to protect the VM and after startup of the VM is complete, a disk benchmarking tool called bonnie++ [16] is run from the VM. Remus and disk TX are then measured.

3.4 Test summary

Although there are five test scenarios described in previous sections, totally nine tests are conducted since 4 of the scenarios are repeated once for Oslo and then again for Gjøvik. Test scenario 5 3.3.5 is only conducted for Gjøvik. A summary of all test scenarios is presented in table 3.7

For all test scenarios, Primary is the main server where the protected VM is started. Remus is then started for that VM from Domain0 on Primary using one of the backups. For example to protect the VM using Gjøvik as backup and using shared storage, following commands are used:

```
#To start the vm
 xm create lenny-vm.cfg
#To start protecting the VM using Remus
 remus --no-net lenny-vm 128.39.80.80
```

In shared storage mode Remus should be started with *–no-net* switch which will not provide fail-over for network connections and no disk updates are sent to backup.

Chapter 4

Results

In this chapter system and network setup along with final results are presented. Including building the virtual machine and setting up the testing environment.

4.1 System and Network setup

Installing and configuring Xen

Operating system used for all servers was Debian Lenny 5.0, which can be downloaded from its website [17]. In Lenny 5.0, two different ways can be used to install Xen, through binary package using packet manager or compile Xen from source that can be downloaded from Xen website [1]. Remus was not part of any Xen official release when this project was started. Only way to get Xen with Remus support was to check out Xen unstable from its repositories and build it manually. The latest official release today is Xen 4.0.0 where Remus is included which was released after these tests were conducted. The detailed information on how to build own Xen virtualization system can be found at the Xen 4.0 Wiki pages [18].

Following support had to be included while building the kernel configuration:

- 802.1d Ethernet Bridging
- IMQ (intermediate queuing device) support
- Universal TUN/TAP device driver support
- Subarchitecture type (Xen-compatible)

Default Dom0 kernel included in Xen is a pvops Linux 2.6.31.x kernel but for this project an older 2.6.18 kernel was used. When system setup is complete, following entry can be created in grub menu list by running *update-grub* command:

```
title           Xen 4.0.0-rc3-pre / Debian GNU/Linux, kernel 2.6.18.8-xen
root            (hd0,0)
kernel          /boot/xen-4.0.0-rc3-pre.gz
module          /boot/vmlinuz-2.6.18.8-xen root=/dev/sda1 ro console=tty0
module          /boot/initrd.img-2.6.18.8-xen
```

After installation Xend can be configured to start at system startup by running following commands:

```
update-rc.d xend defaults 20 21
update-rc.d xendomains defaults 20 21
```

Xend Configuration file

Configuration file for Xend (/etc/xen/xend-config.sxp) was edited for correct setup. Complete Xend configuration file is included in Appendix A.4. Important part of the configuration is the relocation server its port and IP addresses of hosts that are allowed to communicate with Xend.

4.1.1 Virtual Network Configuration

For this project three virtual network segments were created one at each server. A password protected VPN tunnel was established to connect these segments together. A virtual bridge was used at each server using *brctl* tool which is a part of *bridge-utils* package. Commands used to create and configure virtual bridge on Primary:

```
/usr/sbin/brctl addbr virtbr
/sbin/ifconfig virtbr 192.168.0.1 netmask 255.255.255.0 up
```

Same bridge device must exist on each server with IP address from the same private address range used on Primary.

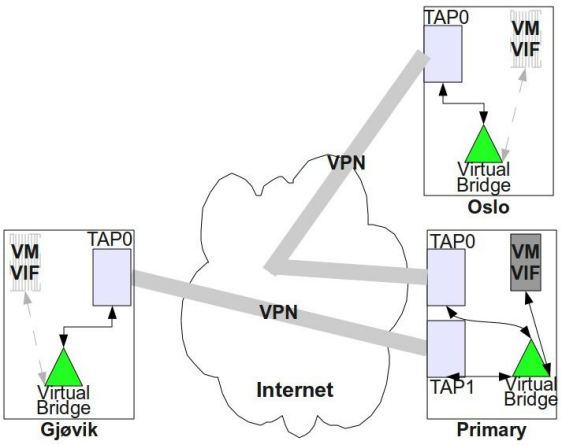

Figure 4.1: **Final virtual network topology**

Final virtual network topology is shown in figure 4.1

4.1.2 Virtual Tunnel VTun

Connecting virtual network segments created in section 4.1.1 was done by establishing VPN tunnels between locations. VTun tool described in section 2.5 was configured with its server running on Primary and Oslo and Gjøvik acting as clients. Complete Vtun server and client configuration files are included

in Appendix A.4. An excerpts follows, showing server side configuration for tunnel between Primary and Oslo:

```
#vtund server configuration file on Primary
                                    .
                                    .
                                    .
remus2_1 {
   passwd 5tyhjmik;     # Password
   type ether;          # IP tunnel
   device tap0;                 #device used
   proto udp;          # TCP protocol

   up {
      ifconfig
        "%% 192.168.0.11 pointopoint 192.168.0.12 mtu 1450";
        program "brctl addif virtbr %%";
   };
   down{
      ifconfig "%% down";
   };
}
                                    .
```

Commands needed for starting Vtun server and Oslo client are listed below:

```
#Vtun server command on Primary
/usr/sbin/vtund -f /etc/vtund-server.conf -m -s

#Command for establishing VPN tunnel between Oslo and Primary on Oslo
/usr/sbin/vtund -f /etc/vtund-client-remus1.conf -m remus2_1 128.39.75.154
```

Server command will instruct vtun to use /etc/vtund-server.conf file as the server configuration and -s switch runs it as a server. According to vtund manual pages, -m switch is useful for NFS traffic which was used in this project. Client side command has in addition the name of the tunnel(remus2_1) on the server along with the server IP address.

4.1.3 VM Creation And Configuration

The VM used for testing was a Debian 5.0 without graphical user interface and was created using debootstrap. Information about how to create a VM filesystem can be found at debian wiki homepage. [19]. Lenny was installed on an ext3 filesystem image.

After VM filesystem creation, additional software was added and network configured. VM filesystem can be mounted and software can be added using packaging tool *apt-get*. Software included is listed in table 3.5. Since VM was to be a part of the virtual network described in section 4.1.1, a private static IP address of **192.168.0.5** was configured for the ethernet interface of the VM and was connected to virtual bridge created in section 4.1.1, when the VM was started.

A copy of the VM filesystem was created on a logical volume on each server for replication. This was done using *dd* command in Linux. The created logical volumes and shared storage paths had to be exactly the same on each server.

Xen Configuration file for the VM

Different Xen configuration files for starting VM were created depending on whether disk was to be replicated or shared storage was to be used. Total of three files were created, one for replicated storage for Oslo, one for Gjøvik and a third one with shared storage which was used for both locations. These are included in Appendix A.2. Kernel used for the VM was same as Dom0 namely 2.6.18.8-Xen. An excerpt of one of the Xen configuration file for the VM follows, showing the kernel, memory and shared storage used:

```
kernel      = '/boot/vmlinuz-2.6.18.8-xen'
ramdisk     = '/boot/initrd.img-2.6.18.8-xen'
memory      = '512'
disk        = ['tap:aio:/mnt/nfs/var/nfs/lenny-1.ext2,xvda1,w']
```

For starting the VM with shared storage following command was used:

```
#Command to start the VM with shared storage on Primary
xm create -c lenny512-remus-nodisk.cfg
```

Remus command used for protecting the VM with shared storage, no disk replicatiopn and no network fail-over:

```
#Remus command for running in shared storage mode
remus --no-net lenny-vm 10.0.0.2
```

Where lenny-vm is the VM name and 10.0.0.2 is the IP address of Oslo backup server.

4.2 Testing Scenario Results

In this section resulting graphs of various test scenarios described in section 3.3 are presented. Results are best presented with graphs and where possible, two graphs are shown in the same axis system. Disk TX traffic to backup when replicated storage is used is shown in separate graphs. Graphs from each test scenario have same time line on the X axis making comparison of graphs easier.

There are five different test scenarios listed in section 3.3 and since four of these scenarios are repeated twice (once for Oslo and then for Gjøvik) a total of nine tests were conducted see table 3.7. For every graph presented X axis is used to represent the time in seconds while Y axis represents traffic in megabytes or kilobytes. For graphs that presents disk writes, the Y axis represents sectors written to disk in the VM. A closer description of the results of these scenarios is given with the results.

4.2.1 Scenario 1: Live migration traffic results

This section presents graphs for measured network traffic load for live migration described in section 3.3.1.

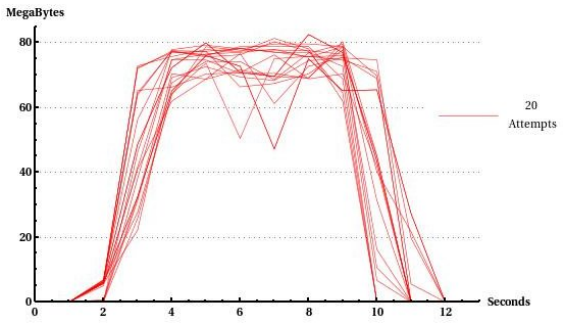

Figure 4.2: **Live migration TX traffic to Oslo**

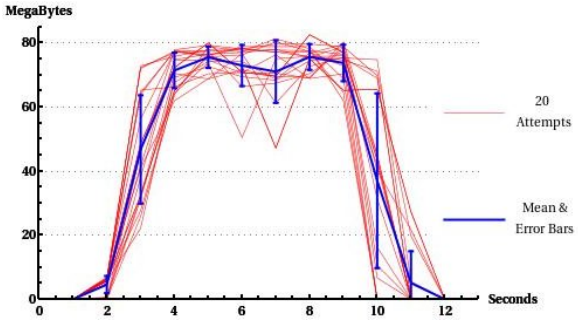

Figure 4.3: **Mean and error bars for all local attempts**

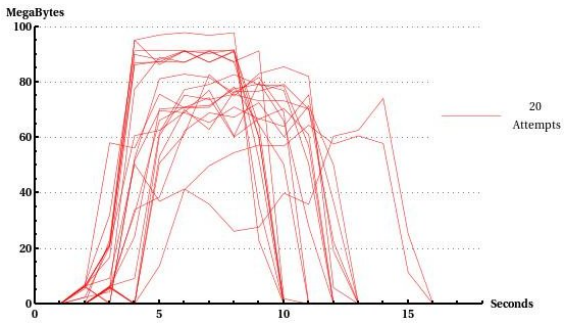

Figure 4.4: **Live migration TX traffic to Gjøvik**

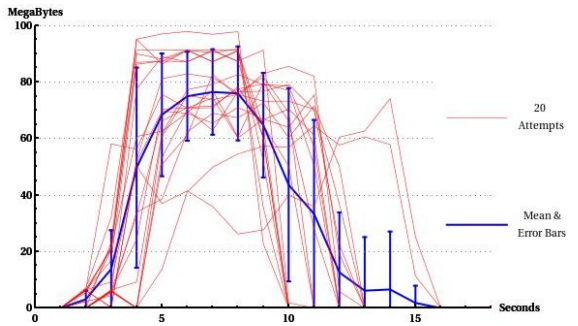

Figure 4.5: **Mean and error bars for all regional attempts**

Figure 4.2 shows traffic created by migrating a VM with resources described in table 3.4 to Oslo while figure 4.3 shows same graphs with mean values of all the attempts and error bars based on standard deviation. Figure 4.4 and 4.5 shows traffic for migrating the VM to Gjøvik. These scenarios were repeated 20 times. Hence the mean and standard deviation values are based on 20 values for each second. For these graphs the X axis represents time in seconds while Y axis represents network traffic in megabytes. On each graph One line shows one migration attempt.

What is seen in figures 4.2 and 4.4 is that from one second into the test, the whole VM memory is transferred to the backup which for Oslo takes between 9 - 10 seconds while for Gjøvik it is between 8 - 15 seconds. It can also be observed from these graphs that local bandwidth is *less* than regional bandwidth available. The maximum transfer rate achieved for Oslo is about 80MB while for Gjøvik it is for some attempts almost 100MB. For a Gigabit link, theoretically 125 Megabyte should be achieved. Comparing graphs for Oslo and Gjøvik, it is also clear that network speeds varies much more for Gjøvik than for Oslo. In this scenario Remus is not involved and it was designed for comparison only. These tests were conducted at different times during 24 hours.

4.3 Scenario 2: Remus network traffic results

As described in scenario 2 3.3.2, network traffic created by running Remus on an idle VM is shown here:

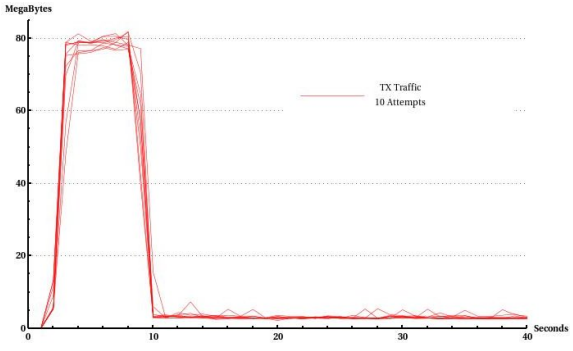

Figure 4.6: **Remus TX traffic for idle VM - local - shared storage**

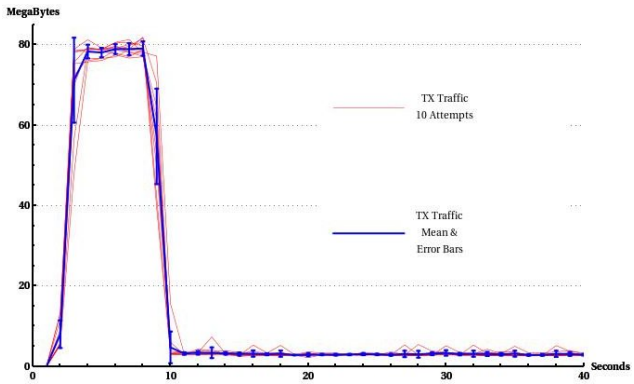

Figure 4.7: **Mean and error bars for figure 4.6**

Figure 4.6 shows network traffic created by running Remus on an Idle VM locally while figure 4.7 includes mean and error bars based on standard deviation for all the attempts in figure 4.6. Same scenario was repeated for Gjøvik, results are shown in figure 4.8 and 4.9. This scenario was repeated 10 times. Therefore mean and standard deviation calculations are based on 10 values for each second. X axis represents time in seconds while Y axis represents network traffic in megabytes.

In this scenario it can be seen from figure 4.6 and figure 4.8 that after whole

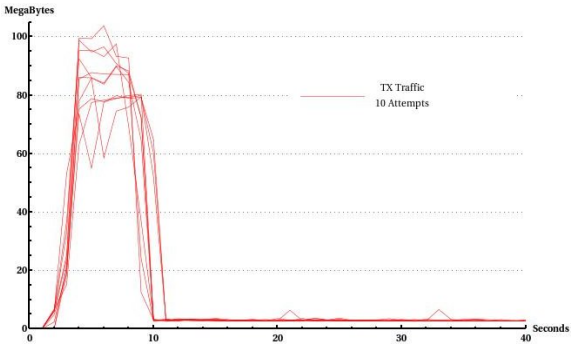

Figure 4.8: **Remus TX traffic for idle VM - regional - shared storage**

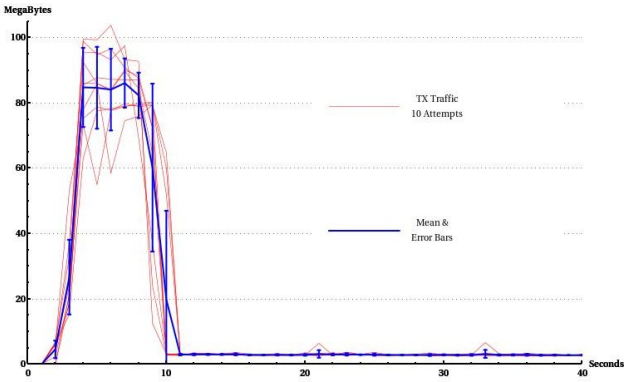

Figure 4.9: **Mean and error bars for figure 4.8**

VM memory transfer, only memory updates are sent to the backup. For an idle VM it is minimum 3 MB per second. This test lasted for 80 seconds and updates after whole memory transfer are shown in figure 4.10 for local and figure 4.11 for regional. Although for some attempts updates to backup varies from 3 MB per second to 7 MB per second, for most attempts the transfer rate is between 3 and 4 MB per second. This is very different from initial memory transfer which is almost 80 MB per second at its peak.

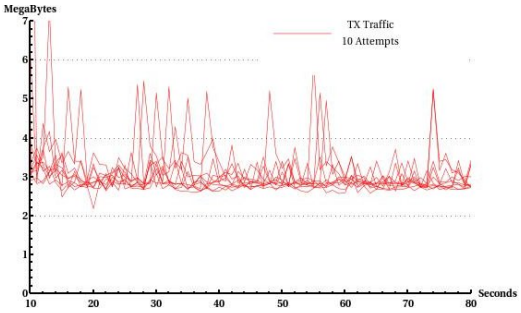

Figure 4.10: **Remus updates to backup for idle VM - local - shared storage**

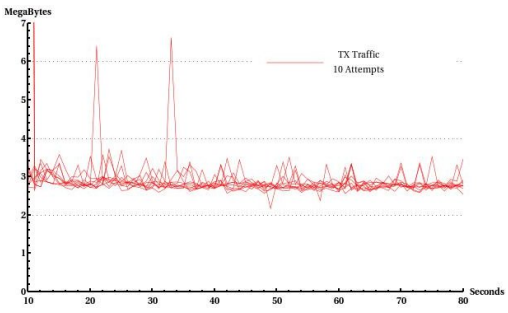

Figure 4.11: **Remus updates to backup for idle VM - regional - shared storage**

Mean and standard deviation calculations

Results presented in figures 4.9 and 4.7 have mean and error bars based on standard deviation. For these results, 20 migration attempts was made therfore the mean and standard deviation calculated for each x value in these graphs, are based on 20 different values.

4.4 Scenario 3: Traffic results of running tests on the VM - No disk replication

In this section results from scenario 3 described in section 3.3.3 are shown where traffic is measured while running different tests on the VM. Shared storage was used so there are no disk updates to the backup.

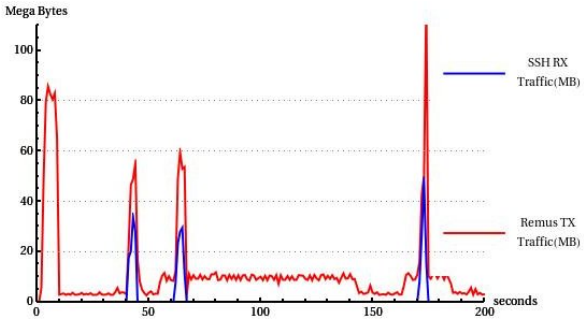

Figure 4.12: **Remus TX and ssh RX traffic - local - shared storage**

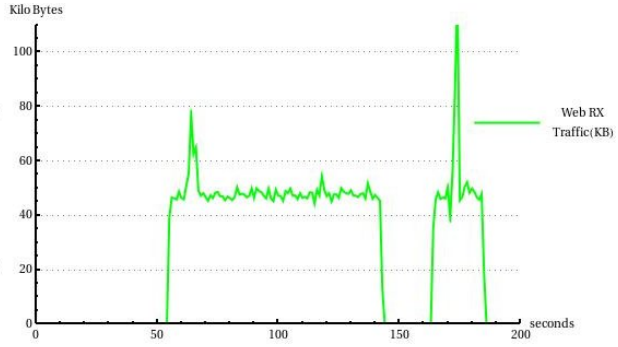

Figure 4.13: **HTTP RX traffic - local - shared storage**

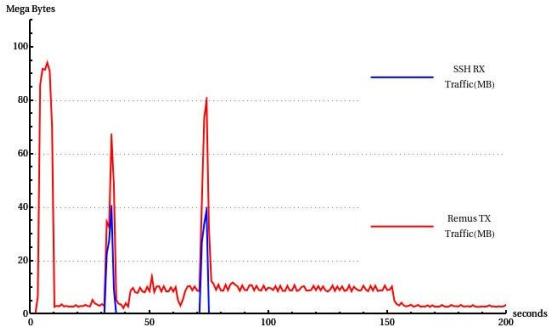

Figure 4.14: **Remus TX and ssh RX traffic - regional - shared storage**

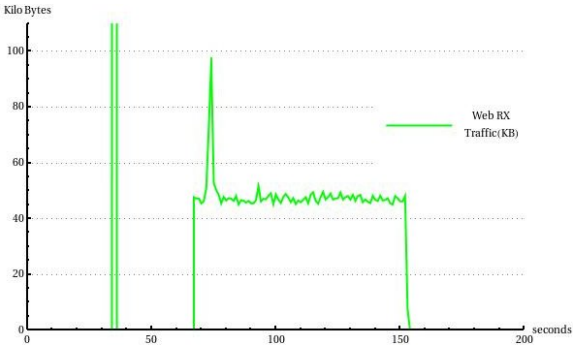

Figure 4.15: **HTTP RX traffic - regional - shared storage**

In figure 4.12, red graph shows Remus traffic to Oslo and blue graph shows SSH traffic to the VM. Figure 4.13 shows web traffic to the VM created by running httperf benchmarking tool. The SSH file transfer in figure 4.12 is repeated 3 times while httperf is run once. Two of the file transfers occurs while httperf is running.

In both figures 4.12 and 4.13 the X axis represents time in seconds while Y axis in figure 4.12 represents traffic in megabytes and kilobytes in figure 4.13. After start, first ssh file transfer is initiated about 40 seconds into the test and it is the only test running at that time. Approximately 54 seconds after start, httperf test is initiated (see figure 4.13) and while this test is still running another ssh file transfer is initiated at 60 seconds. The httperf test is finished after approximately 145 seconds and a new httperf test is initiated approximately 162 seconds into the test and while it is running, 170 seconds

after start, a new file transfer is started as well. The second httperf test is aborted at approximately 188 seconds into the test. Comparing figures for local tests, it can be seen that each time a file transfer is initiated while httperf test is running, a peak occurs in web traffic to the test VM because of retransmission of web connections. The ssh file transfers generates a very high burst of traffic to the backup while httperf test generates a steady amount. The file transfers take between 4-5 seconds while the httperf test should last for 100 seconds, from 54 to 154 seconds from start but is aborted. This whole scenario lasted for almost 190 seconds and is for the local backup server.

For regional test, red graph in figure 4.14 shows Remus traffic to the Gjøvik backup server while blue graph shows ssh file transfer traffic to the test VM. Green graph in figure 4.15 shows web traffic to the test VM created by running httperf tool. Here again X axis for both graphs represents time in seconds while Y axis in figure 4.14 represents traffic in megabytes and kilobytes in figure 4.15. Here SSH file transfer is repeated twice and httperf is run once. The first file transfer is initiated approximately 31 seconds into the test and while this is running, approximately 34 seconds after test is started httperf is initiated towards the test VM. But here since the VM is busy handling file transfer, a burst of web traffic is attempted sent towards the VM which cannot respond in time leading to a lot of retransmission of http traffic. At second file transfer which occurs 71 seconds into the test, httperf is running and there is increased http traffic towards the VM. Here again the file transfers generates high Remus traffic to the backup while httperf generates a steady amount of update traffic to the backup. The whole scenario lasted about 155 seconds and is for regional backup.

To find out how much http and ssh traffic increases the Remus update traffic to backups,figures 4.16 and 4.17 can be studied. These figures shows part of the scenario 3 results. It can be seen from these figures that the amount of Remus traffic created by ssh file transfer is approximately twice the size of ssh traffic. Http traffic to the test VM which is around 50 kilobytes per second, generates about 10 MB per second to the backup,

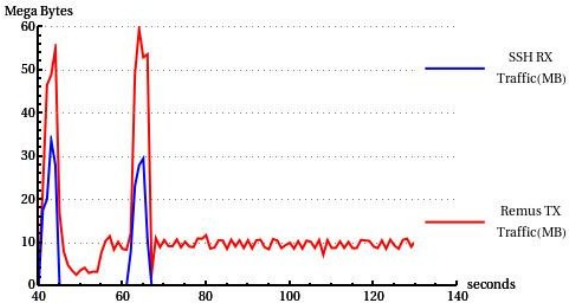

Figure 4.16: **Remus TX and ssh RX comparison - local - shared storage**

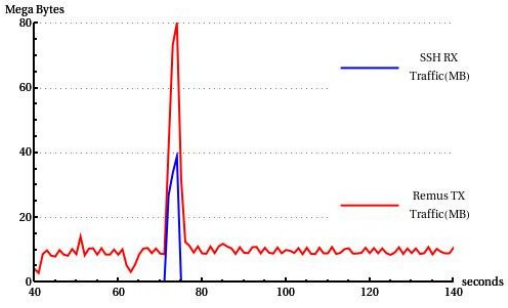

Figure 4.17: **Remus TX and ssh RX comparison - regional - shared storage**

4.5 Scenario 4: Traffic results of running tests on the VM and disk update traffic

Same tests are conducted on the VM and Remus traffic measured in this scenario as in previous scenario 4.4. The only difference here is that Replicated storage is used instead of shared storage. Consequently, in addition to Remus traffic to backup, there are disk updates to backup as well.

In figure 4.18 red graph shows Remus and blue graph shows ssh traffic and Y axis here represents megabytes. The green graph in figure 4.19 shows web traffic created by httperf and Y axis here represents traffic in kilobytes. In figures 4.21 and 4.20 disk traffic to backup in megabytes and sectors written to disk in the VM are shown. All these figures shows traffic to local backup (Oslo).

Here two file transfers are conducted during the scenario that lasted for 90 seconds and two httperf test are started and aborted. The relationship between the ssh -> Remus and http -> Remus traffic is approximately same as in previous scenario. What is different here is that since replicated storage is used here, Remus will open an additional communication channel to the backup where storage write updates are sent to the backup. Therefore each time there is a file transfer to the test VM in figure 4.18 a corresponding update to backup exist in figure 4.21 of same size as the file which is being transferred, in this case 100 MB.

In regional part of the test, there are four file transfers to the test VM and three httperf tests started and aborted in the duration of the test which is about 70 seconds. Ratio between ssh -> Remus and http -> Remus is again approximately same here as previously. But what is interesting here is that in figure 4.22 there are four file transfers to the VM but in figure 4.25 which shows disk updates to backup, there are only two disk updates corresponding to the file transfers. There should have been four disk updates to the backup one for each ssh file transfer to the test VM. This suggests that disk updates to backup have failed after 36 seconds into the test. It is also worth noticing that Remus update traffic to backup continues to function even when disk updates to backup fails.

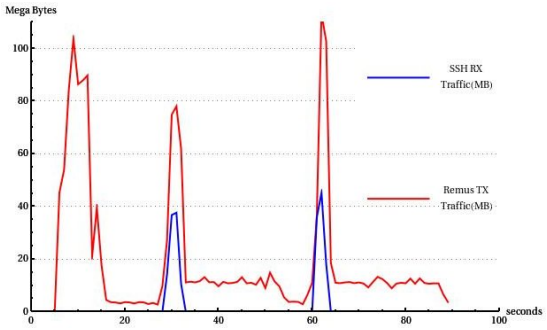

Figure 4.18: **Remus TX and SSH RX traffic - local - replicated storage**

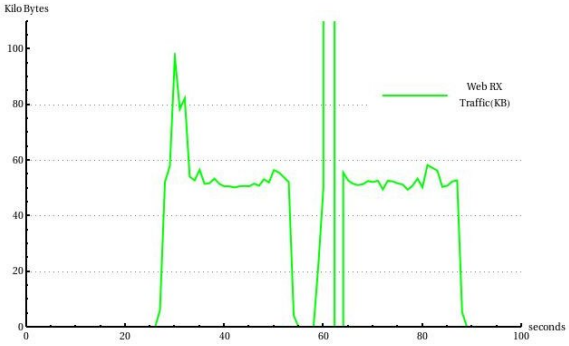

Figure 4.19: **HTTP RX traffic - local - replicated storage**

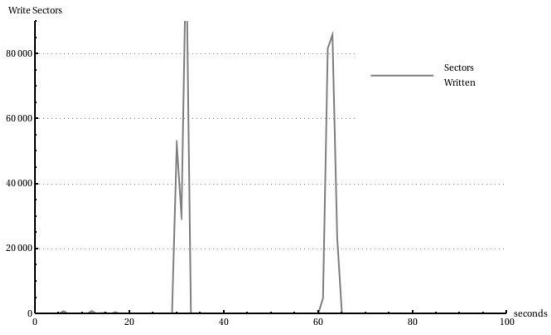

Figure 4.20: **Disk writes in VM - local - replicated storage**

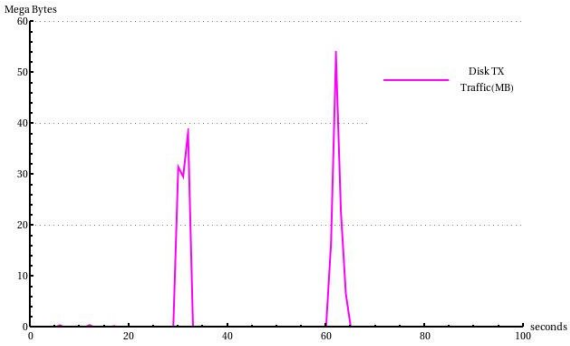

Figure 4.21: **Disk traffic to backup - local - replicated storage**

4.6 Scenario 5: Traffic results of running disk benchmarking tool

Network traffic results for Remus and disk network traffic transmitted to Gjøvik are shown here. In figure 4.26 Remus update traffic along with disk updates in megabyte to backup are shown. While figure 4.27 shows sectors written to disk for the test VM, reported by Xentop, see 3.2.6. After startup, disk benchmarking tool bonnnie++ is started at approximately 56 seconds into the test and lasts for approximately 55 seconds (stops at 111 sec) see figure 4.27. The whole scenario lasted just over 70 seconds. Looking at figure 4.26 representing Remus updates and disk updates to backup, it is clear that both Remus and disk updates failed after just over 70 seconds into the test while the test VM is still operational after this failure. Since the backup VM on Gjøvik server is no longer getting updates from Primary, changes mode to running. This leads to a situation where there are two exactly the same VMs on the network which is not good.

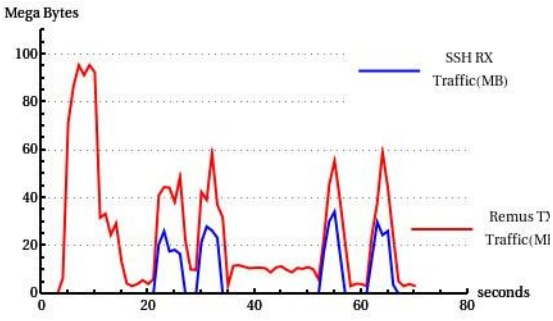

Figure 4.22: **Remus TX and SSH RX traffic - regional - replicated storage**

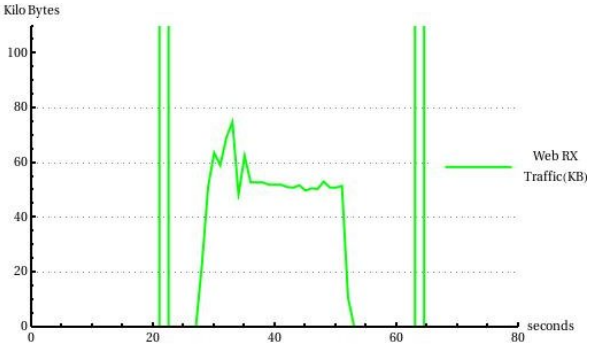

Figure 4.23: **HTTP RX traffic - regional - replicated storage**

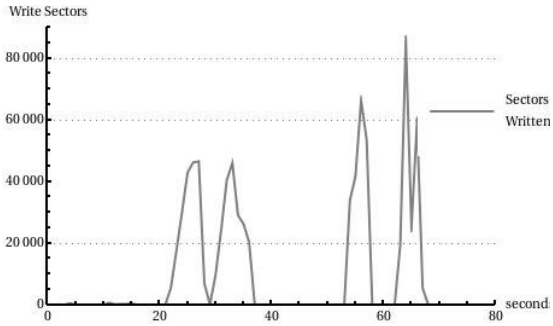

Figure 4.24: **Disk writes in VM - regional - replicated storage**

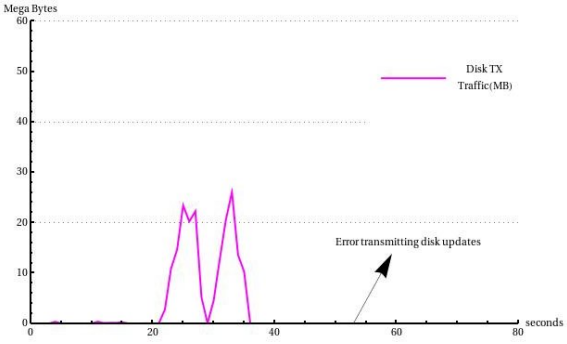

Figure 4.25: **Disk traffic to backup - regional - replicated storage**

4.7 Data Uncertainties

Every data collecting technique have some kind of uncertainty attached to it. Using iptables to collect network traffic is no exception. As described in section 3.2.6, iptables rules were used to collect network traffic for test scenarios in this project. Iptables reports number of packets and bytes for IP packets that matches a rule, therefore if same packet is retransmitted, iptales will report that as new packet every time it is retransmitted. This behavior will introduce uncertainties when measuring network traffic. It is also difficult to calculate how big or large this uncertainty is, but this will be a considerable problem for slow networks. In this project, very high performance networks are used hence it is reasonable to assume that this uncertainty will not have much affect on results in this case.

As described in section 3.2.6, data used as results for various scenarios is network traffic in one direction only. For example for memory updates to backup using Remus, only transmitted traffic is logged responses for this traffic are not included. Same is the case for http and ssh traffic to the VM, only transmitted traffic is logged not responses. This introduces uncertainties in the results presented. But since for all results, network traffic in other direction is much less, the network bandwidth needed will not be affected and therefore

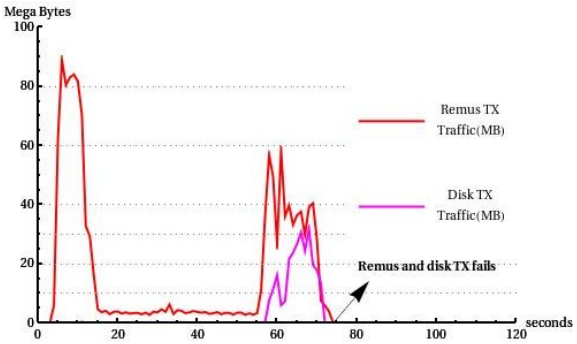

Figure 4.26: **Remus TX and disk TX traffic - regional - replicated storage**

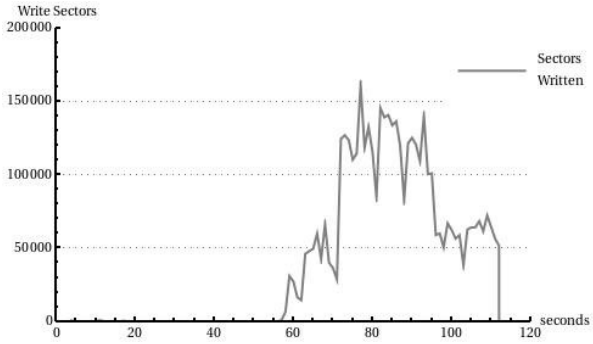

Figure 4.27: **Disk writes in the VM - regional - replicated storage**

this traffic type can be ignored.

As it can be observed from figure 4.26 that initially there was very high network traffic to backup which is similar for all test results(whole VM memory transfer). On the X axis for both figures time in seconds is represented while Y axis in figure 4.27 represents traffic in megabytes and in figure 4.27 it represents sectors written to disk in the VM. After the memory was transferred, update traffic to backup drops and when bonnie++ is started in the VM which is approximately 56 seconds into the test the Remus and disk update traffic to VM goes high again. But there is a problem sending both updates (Remus and disk) to the backup after 75 seconds. From figure 4.27 it is clear that the test VM is still running and writing to disk but Remus has stopped working without changing the test VM mode to suspended. Since updates to backup have stopped the VM at backup will change its mode to running, which means that there are two exactly the same VM on the network and they will most certainly step on each other tows.

Chapter 5

Discussion

Creating Disaster Recovery solutions is hard and solutions are mostly very resource costly and Remus seems to be no exception. Looking at the results from different tests conducted in this project, it is clear that high performance networks are needed for Remus to work satisfactory. The lower the network speed the poorer will be the performance of the VM that is protected using Remus. Even with high network speeds, like in this project, there are other problems that might occur. For instance, from scenario 5 results 4.6 it can be seen that both Remus and disk updates to backup failed even when almost Gigabit network connection existed between Primary and Gjøvik. An explanation for this behavior is given in section 5.4.

5.1 Remus as a disaster recovery system

As mentioned earlier, Remus is now a part of the official release of Xen version 4.0.0 which means that it is accepted as a reliable Disaster Recovery system. Results from this project shows that Remus alone might not be the best solution to use. There has to exist some other logic around Remus if it is used to protect a vital VM. For instance, if used in replicated storage mode, users have to make sure that the disk is always synchronized between Primary and backup. Another problem to look for is the failing of Remus without suspending the Primary VM that would lead to both Primary and backup VM to exist on net which is unacceptable in a production environment. But having said that, Remus has great potential and with further improvements it can be used as a transparent disaster recovery solution where users have access to high performance networks regionally. The statement from the Remus paper quoted under

> Our approach may be used to bring HA "to the masses" as a platform service for virtual machines. [3]

assumes that most people have access to very high performance networks, which is not the case today. But network advancements will in future will make this statement true.

5.2 Remus design

The two mode design solution of Remus is implemented to provide disk replication or only memory synchronization. What is the reason behind implementing disk replication mode as the only mode where network connections fail-over is offered, is not so clear. A guess is that it is just a design decision made by the developers of this tool. Otherwise there is no obvious reason to implement network fail-over in this fashion. Another design feature of Remus is that it can only protect one VM at one time on the same Xen server. This restriction is something that is related to the Xen system.

5.3 Integrity of the system

There is a big concern about integrity of the backup VM which will suffer when Remus is used in replicated storage mode and disk updates to backup fails but memory updates continues to be sent to the backup. This is what happened when running test scenario 4 see figure 4.25. It might be a good idea to do a whole system fail-over when an error of this kind occurs. Because if users are not aware of the fact that disk updates have stopped, a corrupted backup system will take over when disaster happens. In a production environment, this is unacceptable.

Another behavior observed under testing is that if link to backup server is lost for some time and restored, Remus continues to send updates to backup even when some updates are lost when link was down. This will corrupt the backup VM. It is to be mentioned that Remus released with new Xen 4.0.0. is not tested for this behavior.

5.4 Disk solution in Remus - under the hood

For disks or VM filesystems, Remus uses the blktap [20] driver in Xen which is a user level disk IO driver meant to replace the disk loopback driver. Remus design is based on this driver to intercept disk activity from the protected VM. This driver has to be used when running Remus in disk replication mode. Here lies the problem which occurred in test scenario 5 that disk updates to backup suddenly stops. An error message from blktap driver is logged to syslog when disk updates to backup stopped. This happened every time disk benchmarking tool bonnie++ was used in the VM. Therefore, this driver has a weakness which affects Remus in a negative way. In test scenario 4 (see figure 4.25) it also happened that even the Remus memory updates were being sent to the backup, disk updates stopped because of blktap driver problem.

5.5 A different approach of testing Remus

In this project, network traffic generated by Remus is been the focus of testing. But there are other ways that can be used to test this tool. For instance, the pro-

tected VM performance can be the focus of testing which will provide valuable information as well. How different services responses to working under these conditions, their response times compared with running on a non protected VM can be tested. But the main resource needed for Remus to work efficiently is network performance without which all protected VMs will most certainly perform poorly.

5.6 Deciding success factors

In this section, based on this project's results factors that decide whether a regional Disaster Recovery is possible for a system are considered. Which type of processes will generate what type of traffic and how much.

New processes

A key concept in operating systems is the process. A process has its address space, a list of memory locations from 0 to some maximum where the process can read and write. The address space of the process contains the executable program , open files list and list of other related processes. Also all the other information it needs to execute/run is included in its address space.

New processes spawned in the system generates memory changes depending on the type of process. New processes spawned from receiving new connections to the webserver generates less Remus traffic than process that have more disk activity. Specially in the replicated storage mode where same file is transferred to backup first when it is created in memory and again through disk communication channel.

New content Written

Writing new content to disk is a memory consuming process, each byte that is written to disk is first created in memory leading to memory changes at least as large as the content that is to be written. Specially when a file is received for writing to disk through network it is created in the memory each time. Using Remus in disk replication mode, as seen in the results section 4.4, will generate network traffic approximately three times the size of the file that is transferred. Remus update traffic to backup is about twice the size of the file and same file is transferred again through disk communication channel.

Content Read

Content that is read from storage is uploaded to memory thus leading to memory change and for Remus memory updates to backup it will create same amount of traffic as it does for disk writes. But for replicated storage mode, disk reads will not generate any additional traffic on disk communication channel since reads are already transmitted to backup in memory update traffic.

Caching

In operating systems caching is a well known technique for improving performance. Generally, the whole work is done first time and results saved in a cache. For subsequent attempt, cache is checked first and if the result is there it is used otherwise, the whole work is done again. Cache-manager is responsible for caching and it is a continuous process. For Remus caching in the VM means more update traffic to backup since memory changes will occur.

Network IO

For Remus, network traffic to VM will create latency, in disk replication mode, network connections are buffered and external state of VM is not visible until the backup is updated. This will increase round trip time for network traffic to the VM.

5.7 Service types performance

In this section different service types running on the protected VM are considered. How will these services perform running on a VM that is protected using Remus. Services considered are:

1. Game server
2. Mail server
3. Database server

5.7.1 Game server

For a gaming server response times and network speeds are important. All the resources needed for users are attempted allocated first time. A game server wants to avoid disk IO for performance therefore a game server has large application footprint and needs high network speeds. If a game server is run on a VM that is protected using Remus, its performance will depend on what mode Remus is running. In replicated storage mode network latency will increase and game server performance will suffer. In shared storage mode it can work satisfactory since very little disk IO is done by the server.

5.7.2 Mail Server

Mail servers are known for their disk activity, each time a mail is received it is written to disk and read when mail is checked. This will create extensive disk activity which for Remus means more network traffic. Even in shared storage mode Remus traffic be high since disk activity is high. If Remus is used in Replicated storage mode network traffic will increase even more.

5.7.3 Database server

A database server has low network traffic but high disk activity, high caching and high memory usage. This is the worst type of service to use with Remus. Extensive memory changes along with high disk activity will cause very high Remus network traffic for each modes.

Generally, it can be said that services that generates high network traffic but low disk and memory usage will work best with Remus running in shared storage mode. Running Remus in replicated storage mode will increase latency for that service. Services that have extensive disk IO and high memory footprint will have negative impact on Remus update traffic.

Chapter 6

Conclusions

Depending on various test results from this project, some conclusions can be reached which are listed in this section.

High Performance Networks

A very obvious conclusion that can be drawn from this project is that Remus requires very high performance networks to fully protect a VM effectively. This can be observed by reading different graphs from the results section 4.2. The results from test scenario 2 4.3 to 4 4.5 shows that initially Remus will transfer whole memory allocated to the VM, in this case 512MB, before it sends updates of memory changes in the VM to the backup. Results from test scenario 2 4.3 shows that although the VM was idle a constant stream of data was sent to the backup and this was around 3MB per second. This is a high amount of data for an idle VM specially considering the fact that the VM was isolated from all other network traffic and there was no outside interference.

This constant stream of data to backup is heavily dependent on what the VM is doing, the minimum amount of data sent to the backup occurs when the VM is idle. But if VM activity that leads to memory changes occurs, this stream of data will increase depending on how much memory has changed.

Another fact supporting this conclusion is that initially Remus will transfer the whole memory allocated to the VM and the faster it can do this the better will be the performance and respond time of the VM. The VM is suspended every time changed memory is read to be transferred to the backup.

Reason for repeating scenario 1 and scenario 2 many times is to get an fairly accurate idea of the traffic and speeds. Repetitions can be used to calculate means and standard deviations indicating an error margin. It is reasonable to believe that for a setup described in this project these speeds are accurate.

Variance in regional network speeds

Looking at figures 4.8 and 4.6 it is clear that regional network speeds will vary depending on traffic and bandwidth available to Remus will also vary because

of that. For regional test attempts in figure 4.8 maximum speeds reached by each attempt varies much more than for local test attempts in figure 4.6 and these tests were conducted at different times of the day. At day time much more network activity exists on the Universities public network which will result in lower speeds available for Remus. This will affect the performance of the protected VM.

Disk activity versus network traffic

Another observation that can be made by studying results from 4.4 and 4.5 is that disk activity will most certainly generate more traffic than network connections when running Remus in replicated storage mode. Reason is that when a file is transferred to the VM, first this file will be created in memory first [21] and these memory changes will result in increased Remus update traffic to backup. After this the file will be written to the disk which again will generate Remus traffic but now for the disk update channel. For http network connections, a file is asked for for each connection and this file(index.html), is read into memory. There is no disk write and therefore no additional disk update traffic to the backup.

Surprising results

Network speeds are the surprising part of this project, it is tempting to believe that network speeds locally will be higher than speeds for regional network. This is not the case according to figure 4.8 and 4.6. It can be observed from these figures that highest network speeds reached was for the regional setup.

In this project, it is shown that Remus for regional Disaster Recovery can be used if high performance networks are available.

6.1 Future work

In this project, network speeds created by running Remus in different modes is measured. In future work, the test VM can be the focus of attention and it can be tested for performance, latency and behavior. It is also possible to do a similar project with lower regional network speeds while testing the test VM performance and responses. Setting up a test environment using Xen with Remus support is easier now since Remus is a part of official Xen 4.0.0.

Appendix A

A.1 Configuration files

Configuration file for Xen on Primary

```
#
# Xend configuration file.
#
(xend-relocation-server yes)
#(xend-relocation-ssl-server no)
#(xend-udev-event-server no)
# Port xend should use for the relocation interface, if xend-relocation-server
# is set.
(xend-relocation-port 8002)
# Port xend should use for the ssl relocation interface, if
# xend-relocation-ssl-server is set.
#(xend-relocation-ssl-port 8003)
# SSL key and certificate to use for the ssl relocation interface, if
# xend-relocation-ssl-server is set.
# Whether to use ssl as default when relocating.
#(xend-relocation-ssl no)
# Specifying the empty string '' (the default) allows all connections.
(xend-address '')
# Address xend should listen on for relocation-socket connections, if
# xend-relocation-server is set.
(xend-relocation-address '')
(xend-relocation-hosts-allow '')
(network-script network-bridge)
(vif-script vif-bridge)
# dom0-min-mem is the lowest permissible memory level (in MB) for dom0.
# This is a minimum both for auto-ballooning (as enabled by
# enable-dom0-ballooning below) and for xm mem-set when applied to dom0.
(dom0-min-mem 196)
# Whether to enable auto-ballooning of dom0 to allow domUs to be created.
# If enable-dom0-ballooning = no, dom0 will never balloon out.
(enable-dom0-ballooning yes)
(total_available_memory 0)
(dom0-cpus 0)
(vncpasswd '')
```

Configuration file for Vtun server on Primary

```
options {
   port 8006;
   syslog daemon;
   # Path to various programs
   ifconfig      /sbin/ifconfig;
   route         /sbin/route;
   firewall      /sbin/iptables;
   ip            /sbin/ip;
}

# Default session options
default {
   compress no;   # Compression is off
   encrypt no;    # ssh does the encryption
   speed 0;       # By default maximum speed
   keepalive yes;
   stat yes;
}

#Sectiopn for Oslo to Primary
remus2_1 {
   passwd 5tyhjmik;     # Password
   type ether;          # IP tunnel
   device tap0;
   proto udp;          # TCP protocol

   up {
      ifconfig
        "%% 192.168.0.11 pointopoint 192.168.0.12 mtu 1450";
        program "brctl addif virtbr %%";
   };
   down{
      ifconfig "%% down";
   };
}

#Section for GjÃ¸vik to Primary
remus3_1 {
   passwd 5tyhjmik;     # Password
   type ether;          # IP tunnel
   device tap1;
   proto udp;          # TCP protocol

   up {
      ifconfig
        "%% 192.168.0.21 pointopoint 192.168.0.13 mtu 1450";
        program "brctl addif virtbr %%";
   };
   down{
      ifconfig "%% down";
   };
}
```

Configuration file for Vtun client on Oslo

```
options {
   port 8000;
   syslog daemon;
   # Path to various programs
   ifconfig      /sbin/ifconfig;
   route         /sbin/route;
   firewall      /sbin/iptables;
   ip            /sbin/ip;
}

# Default session options
default {
   compress no;   # Compression is off
   encrypt no;    # ssh does the encryption
   speed 0;       # By default maximum speed
   keepalive yes;
   stat yes;
}

#Section for Oslo to Primary
remus2_1 {
   passwd  5tyhjmik;     # Password
   type  ether;          # IP tunnel
   device tap0;
   proto udp;          # TCP protocol

   up {
      ifconfig
        "%% 192.168.0.12 pointopoint 192.168.0.11 mtu 1450";
        program "brctl addif virtbr %%";
   };
   down{
      ifconfig "%% down";
   };
}
```

A.2 Xen configuration files for VM

Xen configuration file for starting VM using Gjøvik as backup with disk replication

```
#
# Configuration file for the Xen instance lenny, created
# by xen-tools 3.9 on Fri Mar 27 20:37:02 2009.
#

#
#  Kernel + memory size
#
kernel      = '/boot/vmlinuz-2.6.18.8-xen'
ramdisk     = '/boot/initrd.img-2.6.18.8-xen'

memory      = '512'

#  Disk device(s).
#
root        = '/dev/xvda1 ro'
disk        = ['tap:remus:128.39.80.80:8000|aio:/dev/xen-domus/lenny-1,xvda1,w']
name        = 'lenny512-lvm'
dhcp        = 'dhcp'
```

```
vif         = [ 'mac=00:16:3E:76:00:AB,bridge=virtbr' ]
on_poweroff = 'destroy'
on_reboot   = 'restart'
on_crash    = 'restart'
extra = '2 console=tty1 xencons=tty'
```

Xen configuration file for starting VM with shared storage

```
#
# Configuration file for the Xen instance lenny, created
# by xen-tools 3.9 on Fri Mar 27 20:37:02 2009.
#

#
#  Kernel + memory size
kernel      = '/boot/vmlinuz-2.6.18.8-xen'
ramdisk     = '/boot/initrd.img-2.6.18.8-xen'
memory      = '512'
#  Disk device(s).
#
root        = '/dev/xvda1 ro'
disk        = ['tap:aio:/mnt/nfs/var/nfs/lenny-1.ext2,xvda1,w']
name        = 'lenny512-lvm'

#  Networking
#
dhcp        = 'dhcp'
vif         = [ 'mac=00:16:3E:76:00:AB,bridge=virtbr' ]
on_poweroff = 'destroy'
on_reboot   = 'restart'
on_crash    = 'restart'
extra = '2 console=tty1 xencons=tty'
```

A.3 Network traffic scripts

Script for measuring Remus network traffic to Gjøvik for replicated storage

```
#!/usr/bin/perl

# This script will first call a perl script for deleting previously created iptables rules if any.
# Then new iptables rules will be created and their packets and bytes logged every second when this scr
# This script must be terminated with Ctrl-c. Each traffic type will be logged to seperate file ending

#some variable definitions
my $ipt="/sbin/iptables";
my $in = "INPUT";
my $out = "OUTPUT";
my $proto = "tcp";
my $vmip = "192.168.0.5";
my $remusip = "128.39.80.80";
my $webport = "80";
my $sshport = "22";
my $diskport = "8000";
my $migport = "8002";
my $target = "ACCEPT";
my ($command,$command1,$command2) = (null,null,null);
my @output = null;


$command = "$ipt -nvxL $out | grep $remusip | grep $migport";
my $result = `$command`;
if($result ne "")
{
```

```
        print "Calling delete script!!\n";
        my @myresult = `/usr/bin/perl ./delete-gjovik-remusNET-web-ssh-ipt-rules.pl`;
}

print "Creating rules for web, ssh and migration...\n";
$command = "$ipt -A $in -p $proto -s $remusip --sport $diskport -j $target";
@output = `$command`;
$command = "$ipt -A $out -p $proto -d $remusip --dport $diskport -j $target";
@output = `$command`;
$command = "$ipt -A $in -p $proto -s $vmip --sport $sshport -j $target";
@output = `$command`;
$command = "$ipt -A $out -p $proto -d $vmip --dport $sshport -j $target";
@output = `$command`;
$command = "$ipt -A $in -p $proto -s $remusip --sport $migport -j $target";
@output = `$command`;
$command = "$ipt -A $out -p $proto -d $remusip --dport $migport -j $target";
@output = `$command`;
$command = "$ipt -A $in -p $proto -s $vmip --sport $webport -j $target";
@output = `$command`;
$command = "$ipt -A $out -p $proto -d $vmip --dport $webport -j $target";
@output = `$command`;


my $remusFILE = "../results/lenny512-gjovik-NET-remus.out";
my $webFILE = "../results/lenny512-gjovik-NET-web.out";
my $sshFILE = "../results/lenny512-gjovik-NET-ssh.out";
my $diskFILE = "../results/lenny512-gjovik-NET-diskusage.out";
my $diskportFILE="../results/lenny512-gjovik-NET-disktraff.out";

open(LOGREMUS," >> $remusFILE") or die "Error: $!\n";
print LOGREMUS "Time\tTX Packets\tRX Packets\tTX Bytes\tRX Bytes\n";

open(LOGWEB," >> $webFILE") or die "Error: $!\n";
print LOGWEB "Time\tTX Packets\tRX Packets\tTX Bytes\tRX Bytes\n";

open(LOGSSH," >> $sshFILE") or die "Error: $!\n";
print LOGSSH "Time\tTX Packets\tRX Packets\tTX Bytes\tRX Bytes\n";

open(LOGDISKUSAGE," >> $diskFILE") or die "Error: $!\n";
print LOGDISKUSAGE "Time\tReads\tWrites\tRsect\tWsect\n";

open(LOGDISKTRAFF," >> $diskportFILE") or die "Error: $!\n";
print LOGDISKTRAFF "Time\tTX Packets\tRX Packets\tTX Bytes\tRX Bytes\n";

my $com_xentop = "xentop -b -x -i 1 |grep RD:";

while ( 1 ){
        my $time = time;
   my $disk_output=`$com_xentop`;   #disk reads and writes
        my ($rem_txp,$rem_rxp,$rem_txb,$rem_rxb) = (0,0,0,0);
        my ($web_txp,$web_rxp,$web_txb,$web_rxb) = (0,0,0,0);
        my ($ssh_txp,$ssh_rxp,$ssh_txb,$ssh_rxb) = (0,0,0,0);
        my ($disk_txp,$disk_rxp,$disk_txb,$disk_rxb) = (0,0,0,0);
   my ($read,$write,$rsect,$wsect)=(0,0,0,0);

        $command = "$ipt -nvxL $out | grep $migport | grep $remusip";
        $result = `$command`;
        $result = substr($result, 0, index($result, 'ACCEPT'));
        @output = ($result =~ m/(\d+) /g);
        $rem_txp = $output[0];
        $rem_txb = $output[1];
        chomp($rem_txp,$rem_txb);

        $command2 = "$ipt -nvxL $in |  grep $migport | grep $remusip";
        $result = `$command2`;
        $result = substr($result, 0, index($result, 'ACCEPT'));
```

```
        @output = ($result =~ m/(\d+) /g);
        $rem_rxp = $output[0];
        $rem_rxb = $output[1];
        chomp($rem_rxp,$rem_rxb);

        my $logremus = "$time\t$rem_txp\t$rem_rxp\t$rem_txb\t$rem_rxb\n";
        print "REMUS:\t\t".$logremus;

        $command1 = "$ipt -nvxL $out | grep $vmip | grep $webport";
        $result = `$command1`;
        $result = substr($result, 0, index($result, 'ACCEPT'));
        @output = ($result =~ m/(\d+) /g);
        $web_txp = $output[0];
        $web_txb = $output[1];
        chomp($web_txp,$web_txb);

        $command = "$ipt -nvxL $in | grep $vmip | grep $webport";
        $result = `$command`;
        $result = substr($result, 0, index($result, 'ACCEPT'));
        @output = ($result =~ m/(\d+) /g);
        $web_rxp = $output[0];
        $web_rxb = $output[1];
        chomp($web_rxp,$web_rxb);

        my $logweb = "$time\t$web_txp\t$web_rxp\t$web_txb\t$web_rxb\n";
        print "WEB:\t\t".$logweb;

        $command2 = "$ipt -nvxL $out |  grep $sshport| grep $vmip";
        $result = `$command2`;
        $result = substr($result, 0, index($result, 'ACCEPT'));
        @output = ($result =~ m/(\d+) /g);
        $ssh_txp = $output[0];
        $ssh_txb = $output[1];
        chomp($ssh_txp,$ssh_txb);

        $command1 = "$ipt -nvxL $in | grep $sshport| grep $vmip";
        $result = `$command1`;
        $result = substr($result, 0, index($result, 'ACCEPT'));
        @output = ($result =~ m/(\d+) /g);
        $ssh_rxp = $output[0];
        $ssh_rxb = $output[1];
        chomp($ssh_rxp,$ssh_rxb);

        my $logssh = "$time\t$ssh_txp\t$ssh_rxp\t$ssh_txb\t$ssh_rxb\n";
        print "SSH:\t\t".$logssh;

    $disk_output =~ m/RD: (.*) WR:/;
    $read = $1;
    $read =~ s/^\s+//; #remove leading spaces
         $read =~ s/\s+$//; #remove trailing spaces

    $disk_output =~ m/WR: (.*) RSECT:/;
    $write = $1;
    $write =~ s/^\s+//; #remove leading spaces
         $write =~ s/\s+$//; #remove trailing spaces

    $disk_output =~ m/RSECT: (.*) WSECT:/;
    $rsect = $1;
    $rsect =~ s/^\s+//; #remove leading spaces
         $rsect =~ s/\s+$//; #remove trailing spaces

    $disk_output =~ m/WSECT: (.*)/;
    $wsect = $1;
    $wsect =~ s/^\s+//; #remove leading spaces
         $wsect =~ s/\s+$//; #remove trailing spaces
```

```
        my $logdiskusage = "$time\t$read\t$write\t$rsect\t$wsect\n";
        print "DISKUSAGE:\t". $logdiskusage;


        $command = "$ipt -nvxL $out | grep $diskport | grep $remusip";
        $result = `$command`;
        $result = substr($result, 0, index($result, 'ACCEPT'));
        @output = ($result =~ m/(\d+) /g);
        $disk_txp = $output[0];
        $disk_txb = $output[1];
        chomp($disk_txp,$disk_txb);

        $command2 = "$ipt -nvxL $in |  grep $diskport | grep $remusip";
        $result = `$command2`;
        $result = substr($result, 0, index($result, 'ACCEPT'));
        @output = ($result =~ m/(\d+) /g);
        $disk_rxp = $output[0];
        $disk_rxb = $output[1];
        chomp($disk_rxp,$disk_rxb);

        my $logdisktraff = "$time\t$disk_txp\t$disk_rxp\t$disk_txb\t$disk_rxb\n";
        print "DISKTRAFF:\t".$logdisktraff;
        print "\n";

        print LOGDISKUSAGE $logdiskusage;
        print LOGDISKTRAFF $logdisktraff;
        print LOGREMUS $logremus;
        print LOGWEB $logweb;
        print LOGSSH $logssh;
        sleep 1;
}
```

Script for deleting iptables rules for Gjøvik replicated storage

```
#!/usr/bin/perl
#some variable definitions
my $ipt="/sbin/iptables";
my $in = "INPUT";
my $out = "OUTPUT";
my $proto = "tcp";
my $vmip = "192.168.0.5";
my $remusip = "128.39.80.80";
my $webport = "80";
my $sshport = "22";
my $diskport = "8000";
my $migport = "8002";
my $target = "ACCEPT";
my $command = null;
my @output = null;

#Reset input output chanins...
$command = "$ipt -Z $in";
@output = `$command`;
$command = "$ipt -Z $out";
@output = `$command`;
print "Deleting iptables rules for web, ssh and xen...\n";
$command = "$ipt -D $in -p $proto -s $vmip --sport $sshport  -j $target";
@output = `$command`;
$command = "$ipt -D $out -p $proto -d $vmip --dport $sshport  -j $target";
@output = `$command`;
$command = "$ipt -D $in -p $proto -s $remusip --sport $migport  -j $target";
@output = `$command`;
$command = "$ipt -D $out -p $proto -d $remusip --dport $migport  -j $target";
@output = `$command`;
$command = "$ipt -D $in -p $proto -s $vmip --sport $webport  -j $target";
@output = `$command`;
```

```
$command = "$ipt -D $out -p $proto -d $vmip --dport $webport  -j $target";
@output = `$command`;
$command = "$ipt -D $in -p $proto -s $remusip --sport $diskport  -j $target";
@output = `$command`;
$command = "$ipt -D $out -p $proto -d $remusip --dport $diskport  -j $target";
@output = `$command`;


exit 0;
```

Script for converting and dividing traffic files for use with Mathematica

```
#!/usr/bin/perl

#
# This script will read all files .out files from a directory and divide them into attempts caculating
# right values for each entry. A file for use for statistics will also be created for all .out files.
#
#use bignum ( p => 50 );
my $dir = "/experiments/results";
opendir(DIR, $dir) or die "can't opendir $dir: $!";
my @files = grep {  $_ ne '.' && $_ ne '..' && $_ ne "parts" && !/~/ && $_ ne 'ping-times' && !/pl/} re
foreach $file (@files)
{
    # do something with "$dirname/$file"
 my $num = 1;
 my $lines = 1;
 my $infile = "$dir/$file";
 my $outfile = "$dir/parts/".substr($file, 0, index($file, 'out')).$num;
 my $allfile = "$dir/parts/".substr($file, 0, index($file, 'out'))."all";
 my $statfile = "$dir/parts/".substr($file, 0, index($file, 'out'))."stat";
 my @raw_times = NULL;
 my @real_times = NULL;
 my $k = 0;
 my $kilo = 1024;
 my $mega = 1048576;

 if ($ARGV[0] ne ""){ $infile = $ARGV[0]}

 open(IN, "<$infile") or die "USAGE: $0 INPUTFILE \nError: $!";
 open(OUT,">$outfile") or die "USAGE: $0 INPUTFILE \nError: $!";
 open(ALL,">$allfile") or die "USAGE: $0 INPUTFILE \nError: $!";
 open(STAT,">$statfile") or die "USAGE: $0 INPUTFILE \nError: $!";

 my @raw_data = <IN>;
 my @diff_data = NULL;
 my ($time, $tx_packets, $rx_packets, $tx_bytes, $rx_bytes) =(0,0,0,0,0);
 my ($time1, $time2, $tx_packets1, $tx_packets2, $rx_packets1, $rx_packets2, $tx_bytes1, $tx_bytes2, $r
 my ($tot_tx_packets, $tot_rx_packets, $tot_tx_bytes, $tot_rx_bytes) = (0,0,0,0);
 $k = 0;
 print OUT "Attempts\tTX-PACKETS\tRX-PACKETS\tTX-BYTES\tRX-BYTES\n";
 $diff_data[0] = "Time\tTX PACKETS\tRX PACKETS\tTX BYTES\tRX BYTES\n";

 for(my $i=1; $i<=scalar@raw_data; $i++)
 {
  ($time1, $tx_packets1, $rx_packets1, $tx_bytes1, $rx_bytes1) = (0,0,0,0,0);
  ($time2, $tx_packets2, $rx_packets2, $tx_bytes2, $rx_bytes2) = (0,0,0,0,0);
  ($time, $tx_packets, $rx_packets, $tx_bytes, $rx_bytes) = (0,0,0,0,0);
  ($time1, $tx_packets1, $rx_packets1, $tx_bytes1, $rx_bytes1) = split('\t',$raw_data[$i]);
  if(scalar@raw_data > $i+1)
  {
   ($time2, $tx_packets2, $rx_packets2, $tx_bytes2, $rx_bytes2) = split("\t",$raw_data[$i+1]);
   if ($time2 =~ /^-?\d/) #If Time is a number or its the first row
   {
    $time = $time2-$time1;
    $tx_packets=$tx_packets2-$tx_packets1;
```

```
    $rx_packets = $rx_packets2-$rx_packets1;
    $tx_bytes=$tx_bytes2-$tx_bytes1;
    $rx_bytes=$rx_bytes2-$rx_bytes1;
    $diff_data[$i] = "$time\t$tx_packets\t$rx_packets\t$tx_bytes\t$rx_bytes\n";
      print OUT "$lines\t$tx_packets\t$rx_packets\t$tx_bytes\t$rx_bytes\n";
      $lines += 1;
      $tot_tx_packets += $tx_packets;
      $tot_rx_packets += $rx_packets;
      $tot_tx_bytes += $tx_bytes;
      $tot_rx_bytes += $rx_bytes;
   }
   else{ #print OUT "$time2\t$tx_packets2\t$rx_packets2\t$tx_bytes2\t$rx_bytes2\n";
      $diff_data[$i] = "$time2\t$tx_packets2\t$rx_packets2\t$tx_bytes2\t$rx_bytes2";
      close OUT;
      $num += 1;
      $outfile = "$dir/parts/".substr($file, 0, index($file, 'out')).$num;
      open(OUT,">$outfile") or die "USAGE: $0 INPUTFILE \nError: $!";
      print OUT "Attempts\tTX-PACKETS\tRX-PACKETS\tTX-BYTES\tRX-BYTES\n";
      $i=$i+1;
      $lines = 1;
      #print OUT "$lines\t$tot_tx_packets\t$tot_rx_packets\t$tot_tx_bytes\t$tot_rx_bytes\n";
      ($tot_tx_packets, $tot_rx_packets, $tot_tx_bytes, $tot_rx_bytes) = (0,0,0,0);
     }
 }
}
 #print DIFF @diff_data;

#Now create the file for statistic use
$row=1;
$col=1;
my @part=();
for(my $i=1; $i<scalar@diff_data; $i++)
{
 if($diff_data[$i] =~ m/TX/)
 {
  $col++;
  $row=0;
 }
 else {
  $part[$col][$row]= $diff_data[$i];
  $row++;
 }
}
my $leastentries=100000000;
my @result=null;
my $head="Sec\t";
print "#part IS $#part\n";
for $i ( 1 .. $#part ) {
 $head = $head."Test$i\t";
 for $j ( 1 .. $#{$part[$i]} ) {
  if($#{$part[$i]} < $leastentries){$leastentries=$#{$part[$i]};}
 }
}
$head = $head."Mean\tStDev";
for $i ( 1 .. $leastentries ) {
 my $tot_txb = 0;
 my $mean = 0;
 my $stddev = 0;
 my $sum = 0;
 my @tx_values = NULL;
 for $j ( 1 .. $#part ) {
  ($time, $tx_packets, $rx_packets, $tx_bytes, $rx_bytes) = split("\t",$part[$j][$i]);
  $result[$i] = $result[$i].$tx_bytes."\t"; #add a comma at the end here if comma seperated is needed
  $tot_txb = $tot_txb + $tx_bytes;
   }
 #caculate mean value
```

```
  $mean=$tot_txb/$#part;
  @tx_values = split("\t",$result[$i]);
  for(my $x=0; $x<scalar@tx_values; $x++){
   $sum = $sum + ($tx_values[$x]-$mean);
  }
  $sum = $sum**2;
  if(scalar@tx_values-1 != 0){$stddev = sqrt($sum/(scalar@tx_values-1));}
  else {$stddev = 0;}
  $result[$i] = $result[$i]."$mean\t$stddev";
  }
 print STAT $head."\n";
 for(my $i=1; $i<scalar@result; $i++){
  print STAT "$i\t$result[$i]\n";
 }
 print ALL @diff_data;
 close ALL;
 close IN;
 close OUT;
 close STAT;
}
closedir(DIR);
exit 0;
```

A.4 Network traffic log files

An example network traffic log file

Time	TX Packets	RX Packets	TX Bytes	RX Bytes
1272324555	0	0	0	
1272324556	0	0	0	
1272324558	0	0	0	
1272324559	26	10	31584	544
1272324560	4545	1839	6798556	95652
1272324561	54402	17113	81567516	889900
1272324563	114921	35900	172330748	1879388
1272324564	181617	54895	272355788	2867128
1272324565	245429	72920	368056928	3804428
1272324567	312157	93464	468127724	4872716
1272324568	376805	108883	565076848	5674504
1272324569	399010	112198	598380376	5846884
1272324570	422345	114171	633378976	5949480
1272324572	439546	116116	659177044	6050620
1272324573	460077	118293	689968488	6163824
1272324574	469554	118946	704178352	6197780
1272324575	472576	119283	708706840	6215304
1272324576	474810	119584	712054428	6230956
1272324577	477669	119944	716336640	6249676
1272324579	481624	120270	722266520	6266628
1272324580	484507	120640	726586880	6285868
1272324581	488707	121004	732882224	6304796
1272324582	517399	124197	775916056	6470832
1272324583	548508	126295	822576976	6579928
1272324585	579483	126931	869034248	6613000
1272324586	606255	127094	909191248	6621476
1272324587	640366	127919	960354668	6670640
1272324589	655985	128266	983779324	6688684
1272324590	663091	128837	994432380	6718376
1272324591	670028	129915	1004834748	6774432
1272324592	699732	131655	1049385588	6864912
1272324594	727110	131958	1090449248	6880668
1272324595	768212	132544	1152095520	6911140
1272324597	794177	135060	1191037348	7041972
1272324598	816452	138995	1224446768	7246592

1272324600	818826	139045	1228006320	7249965
1272324601	826986	139732	1240246320	7347331
1272324602	835334	140371	1252764696	7422361
1272324603	843239	140544	1264619600	7431357
1272324605	850702	140624	1275811920	7435517
1272324606	858262	140707	1287148684	7439833
1272324607	865879	140785	1298570412	7443889
1272324608	873323	140962	1309733044	7453093
1272324610	879558	141130	1319081984	7461829
1272324611	887214	141366	1330563140	7474101
1272324612	895220	141575	1342566796	7484969
1272324613	902210	141772	1353048264	7495213
1272324614	908469	141945	1362432808	7504209
1272324616	915993	142170	1373714400	7515909
1272324617	923287	142357	1384651932	7525633
1272324618	931079	142576	1396336960	7537021
1272324619	938229	142775	1407057904	7547369
1272324620	942389	142895	1413293768	7553609
1272324622	958673	143966	1437718076	7609301
1272324623	990481	145687	1485425824	7698793
1272324625	1029284	148655	1543630048	7853129
1272324626	1056802	150243	1584905484	7935705
1272324627	1071569	151227	1607052144	7986873
1272324628	1073795	151291	1610387016	7990201
1272324629	1076711	151376	1614754192	7994621
1272324630	1079581	151477	1619053376	7999873
1272324632	1081838	151545	1622434960	8003409
1272324633	1098978	152326	1648141012	8044021
1272324634	1125343	153346	1687682824	8097061
1272324636	1166767	154804	1749812300	8172877
1272324637	1197682	156418	1796180600	8262337
1272324638	1214574	157135	1821514452	8299621
1272324639	1218206	157245	1826956160	8305341
1272324641	1220421	157313	1830274060	8308877
1272324642	1223271	157400	1834543504	8313401
1272324643	1225596	157470	1838027024	8317041

An example network traffic log file converted for drawing graphs

Time	TX Packets	RX Packets	TX Bytes	RX Bytes
1272324555	0	0	0	
1272324556	0	0	0	
1272324558	0	0	0	
1272324559	26	10	31584	544
1272324560	4545	1839	6798556	95652
1272324561	54402	17113	81567516	889900
1272324563	114921	35900	172330748	1879388
1272324564	181617	54895	272355788	2867128
1272324565	245429	72920	368056928	3804428
1272324567	312157	93464	468127724	4872716
1272324568	376805	108883	565076848	5674504
1272324569	399010	112198	598380376	5846884
1272324570	422345	114171	633378976	5949480
1272324572	439546	116116	659177044	6050620
1272324573	460077	118293	689968488	6163824
1272324574	469554	118946	704178352	6197780
1272324575	472576	119283	708706840	6215304
1272324576	474810	119584	712054428	6230956
1272324577	477669	119944	716336640	6249676
1272324579	481624	120270	722266520	6266628
1272324580	484507	120640	726586880	6285868
1272324581	488707	121004	732882224	6304796
1272324582	517399	124197	775916056	6470832
1272324583	548508	126295	822576976	6579928
1272324585	579483	126931	869034248	6613000
1272324586	606255	127094	909191248	6621476

1272324587	640366	127919	960354668	6670640
1272324589	655985	128266	983779324	6688684
1272324590	663091	128837	994432380	6718376
1272324591	670028	129915	1004834748	6774432
1272324592	699732	131655	1049385588	6864912
1272324594	727110	131958	1090449248	6880668
1272324595	768212	132544	1152095520	6911140
1272324597	794177	135060	1191037348	7041972
1272324598	816452	138995	1224446768	7246592
1272324600	818826	139045	1228006320	7249965
1272324601	826986	139732	1240246320	7347331
1272324602	835334	140371	1252764696	7422361
1272324603	843239	140544	1264619600	7431357
1272324605	850702	140624	1275811920	7435517
1272324606	858262	140707	1287148684	7439833
1272324607	865879	140785	1298570412	7443889
1272324608	873323	140962	1309733044	7453093
1272324610	879558	141130	1319081984	7461829
1272324611	887214	141366	1330563140	7474101
1272324612	895220	141575	1342566796	7484969
1272324613	902210	141772	1353048264	7495213
1272324614	908469	141945	1362432808	7504209
1272324616	915993	142170	1373714400	7515909
1272324617	923287	142357	1384651932	7525633
1272324618	931079	142576	1396336960	7537021
1272324619	938229	142775	1407057904	7547369
1272324620	942389	142895	1413293768	7553609
1272324622	958673	143966	1437718076	7609301
1272324623	990481	145687	1485425824	7698793
1272324625	1029284	148655	1543630048	7853129
1272324626	1056802	150243	1584905484	7935705
1272324627	1071569	151227	1607052144	7986873
1272324628	1073795	151291	1610387016	7990201
1272324629	1076711	151376	1614754192	7994621
1272324630	1079581	151477	1619053376	7999873
1272324632	1081838	151545	1622434960	8003409
1272324633	1098978	152326	1648141012	8044021
1272324634	1125343	153346	1687682824	8097061
1272324636	1166767	154804	1749812300	8172877
1272324637	1197682	156418	1796180600	8262337
1272324638	1214574	157135	1821514452	8299621
1272324639	1218206	157245	1826956160	8305341
1272324641	1220421	157313	1830274060	8308877
1272324642	1223271	157400	1834543504	8313401
1272324643	1225596	157470	1838027024	8317041

Bibliography

[1] XEN, 2009. http://www.xen.org/.

[2] Brett J. L. Landry and M. Scott Koger. Dispelling 10 common disaster recovery myths: Lessons learned from hurricane katrina and other disasters. *J. Educ. Resour. Comput.*, 6(4):6, 2006.

[3] Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield. Remus: high availability via asynchronous virtual machine replication. In *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161–174, Berkeley, CA, USA, 2008. USENIX Association.

[4] Daniel Bartholomew. Getting started with heartbeat. *Linux J.*, 2007(163):2, 2007.

[5] Th. Lumpp, J. Schneider, J Holtz, M. Mueller, N. Lenz, A. Biazetti, and D. Petersen. From high availability and disaster recovery to business continuity solutions. *IBM Syst. J.*, 47(4):605–619, 2008.

[6] Pedro Pla. Drbd in a heartbeat. *Linux J.*, 2006(149):3, 2006.

[7] Christopher Strachey. Time sharing in large fast computers. *Proceedings of the International Conference on Information processing*, 1959.

[8] Kirk L. Kroeker. The evolution of virtualization. *Commun. ACM*, 52(3):18–20, 2009.

[9] Reinhold KrÂ¨oger Dan Marinescu. State of the art in autonomic computing and virtualization, 2007. http://wwwvs.cs.hs-rm.de/downloads/extern/pubs/techreports/STAR.pdf.

[10] Haikun Liu, Hai Jin, Xiaofei Liao, Liting Hu, and Chen Yu. Live migration of virtual machine based on full system trace and replay. In *HPDC '09: Proceedings of the 18th ACM international symposium on High performance distributed computing*, pages 101–110, New York, NY, USA, 2009. ACM.

[11] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[12] Maxim Krasnyansky. Vtun-virtual tunnels over tcp/ip networks, 1999. http://vtun.sourceforge.net/.

[13] Netfilter.org. The netfilter.org iptables project., 2010. http://netfilter.org/projects/iptables/index.html.

[14] unixref.com. Xentop - display real time info about xen, 2010. http://www.unixref.com/manPages/xentop.html.

[15] L.P. Hewlett-Packard Development Company. Welcome to the httperf homepage, 2009. http://www.hpl.hp.com/research/linux/httperf/.

[16] Tim Bray and Russel Coker. Bonnie++ documentation, 1999. http://www.coker.com.au/bonnie++/readme.html.

[17] debian.org. Debian - the universal operating system, 2010. http://www.debian.org/.

[18] xen.org. Xen 4.0 - wiki, 2010. http://wiki.xensource.com/xenwiki/Xen4.0.

[19] debianWiki. Debian - wiki, 2010. http://wiki.debian.org/xen.

[20] Andrew Warfield and Julian Chesterfield. blktap - xen wiki, 2008. http://wiki.xensource.com/xenwiki/blktap.

[21] Andrew S. Tanenbaum. *Modern Operating Systems.*, pages 275–278. Pearson Education Inc., Chicago, Illinois, United States of America, third edition edition, 2009.